Министерство образования Российской Федерации

Санкт-Петербургский
Государственный Технический Университет

Центральный научно-исследовательский
и опытно-конструкторский институт
робототехники и технической кибернетики

В.С. Заборовский

СЕТИ ЭВМ И ТЕЛЕКОММУНИКАЦИИ

Моделирование и анализ сетей связи с коммутацией пакетов

Network Simulator (Сетевой симулятор ns2)

Учебное пособие

Санкт-Петербург
Издательство СПбГТУ
2001

УДК 061.68

Анфилатов Ю.В., Заборовский В.С., Подгурский Ю.Е. **Сети ЭВМ и телекоммуникации. Моделирование и анализ сетей связи с коммутацией пакетов.** Network Simulator (Сетевой симулятор ns2): Учеб. пособие. СПб.: Изд-во СПбГТУ, 2001, с.

Пособие соответствует дисциплине ОПД Ф 10 “Сети ЭВМ и телекоммуникации” и дисциплине СДМ магистерской программы 552813 “Сети ЭВМ и телекоммуникации” государственного образовательного стандарта направления 552800 “Информатика и вычислительная техника”.

В пособии дано краткое описание сетевого симулятора NS2 (Network Simulator, версия 2) - программной событийной модели, широко используемой для исследования функционирования сетей с коммутацией пакетов. Изложены методы построения имитационных моделей для исследования процессов в компьютерных сетях на различных уровнях взаимодействия. Приведены методические материалы к циклу лабораторных НИР студентов, разработанные на кафедре телематики СПбГТУ

Пособие предназначено для студентов старших курсов специальностей “Информатика и вычислительная техника”, “Сети ЭВМ и телекоммуникации”, а также для специалистов, интересующихся современными методами анализа функционирования сетей связи с коммутацией пакетов.

Ил. 20 Библиогр.: 5 назв.

СОДЕРЖАНИЕ

СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

ATM	Asynchronous Transfer Mode
CBR	Constant Bit Rate
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CWND	Congestion Window
DM	Dense Mode
DV	Distance Vector (routing)
ENET	Ethernet
FDDI	Fibre Distributed Data Interface
FIFO	First-In-First-Out
FTP	File Transfer Protocol
HFC	Hybrid Fiber Coaxial (гибридные оптокоаксиальные сети кабельного TV)
IP	Internet Protocol
ISO/OSI	International Standards Organization / Open System Interconnection
ITU	International Telecommunication Union
LAN	Local Area Network
LL	Link Layer
MAC	Medium Access Control
MSS	Maximum Segment Size -
Nam	Network Animator
NS	Network Simulator
PC	Personal Computer
PDU	Protocol Data Unit
PHY	Physical Layer
QoS	Quality of Service
RTO	Retransmission TimeOut
RTT	Round Trip Time
SACK	Selected Acknowledgement
SNMP	Simple Network Management Protocol
TCL	Tool Command Language
TCP	Transmission Control Protocol
TR	Token Ring
UDP	User Datagram Protocol
VC	Virtual Channel

АП	абонентский пункт
АТС	автоматическая телефонная станция
БД	база данных
ВЛВС	виртуальная ЛВС
ВОС	взаимодействие открытых систем
ИВС	информационно-вычислительная система
КТВ	кабельное телевидение
ЛВС	локальная вычислительная сеть
ОС	операционная система
ПК	персональный компьютер

ВВЕДЕНИЕ

Обеспечение эффективного использования ресурсов компьютерных сетей, на фоне постоянного усложнения их топологии, появления новых сетевых технологий, протоколов и приложений, требует использования разнообразных средств моделирования и анализа функционирования сетевых структур. Отметим, что современные компьютерные сети представляют собой весьма сложный объект исследования: в силу их естественной распределенности и децентрализации, проведение "натурных" экспериментов крайне затруднено, поэтому моделирование является практически единственным методом изучения сетевых процессов и исследования свойств новых протоколов.

В настоящее время известно большое число разнообразных средств сетевого моделирования. Наиболее распространенными являются следующие программные продукты: PlanNet фирмы Comdisco, NetMaker (MakeSystems), NETWORK II.5 (CASI), Optimal Performance, CoreNet Management System, X-kernel (Университет шт. Аризона) и др. Все они различаются назначением, возможностями и стоимостью (от бесплатных до $ 50000). Сравнительный анализ некоторых из них приведен в [1].

В данном пособии дано краткое описание сетевого симулятора NS2 (Network Simulator, версия 2), разработанного в Калифорнийском университете США, - программной событийной модели, широко используемой для исследования процессов в компьютерных сетях. Помимо описания основных компонентов симулятора и принципов моделирования, в пособии приведены методические материалы к циклу лабораторных НИР студентов, разработанные на кафедре телематики СПбГТУ. В основу цикла НИР положен принцип поэтапного перехода от изучения отдельных возможностей симулятора к освоению принципов построения моделей для целенаправленного исследования реальных сетевых структур.

1. Сетевой симулятор NS2 (Network Simulator -2)

1.1 Назначение

Сетевой симулятор NS2 (далее, симулятор) представляет собой программное средство для моделирования и анализа функционирования цифровых сетей с коммутацией пакетов. Первая версия симулятора появилась в 1989 году, когда быстрое распространение технологии межсетевого взаимодействия (internetworking) вызвало необходимость исследования и совершенствования протоколов составных сетей. Широкие возможности симулятора для исследования корректности и эффективности протоколов различных уровней, а также для моделирования разнородных приложений способствовали его быстрому распространению. С 1995 года совершенствование симулятора поддерживается агентством перспективных исследований министерства обороны (DARPA) США в рамках проектов VINT (Virtual InterNetwork Testbed) и SAMAN (Simulation Augmented by Measurement and Analysis for Networks), а также национальным научным фондом США (проект CONSER – Collaborative Simulation for Education and Research).

Симулятор предназначен для использования, как в исследовательских целях – для оценки влияния различных факторов на эффективность разрабатываемых протоколов и приложений, так и в учебных целях - для пояснения работы конкретных протоколов и алгоритмов управления процессами в сетях.

В зависимости от целей исследования, процессы в сети могут моделироваться на различных уровнях взаимодействия, с учетом особенностей как традиционных, так и перспективных приложений, протоколов и технологий (WWW, Multicast, Mobile Networking, Satellite Networking, LAN и др.).

Основными особенностями симулятора NS2 являются:

- **Модульный принцип построения и открытая архитектура** симулятора, соответствующие многоуровневой архитектуре сетей и позволяющие легко расширять его функциональные возможности путем добавления новых модулей и модификации имеющихся.
- **Широкий диапазон методов и средств абстрагирования,** предоставляющих возможности изменения уровня абстракции, как при анализе результатов, так и, непосредственно, при моделировании. Это позволяет идентифицировать существенные эффекты на уровне макромодели, а затем применять детальное моделирование для их более тщательного исследования.

- **Наличие средств анимации,** обеспечивающих возможность наблюдения конкретных реализаций и позволяющих выделить наиболее важные и интересные моменты поведения сетевой модели. (Отметим, что при рассмотрении только агрегированных статистических оценок много существенных явлений остается незамеченными).
- **Наличие библиотеки сетевых топологий и генераторов трафика,** облегчающих создание моделей сетей со сложной топологией и смешанной нагрузкой.

  Состав смешанного Интернет-трафика постоянно меняется (в настоящее время преобладают WWW-приложения, в ближайшем будущем, возможно, основную долю составят аудио и видеоприложения). Топология сетей также постоянно изменяется. Важно облегчить исследователям возможность опережающего исследования влияния этих тенденций на эффективность работы сетей. Проблема задания топологии и динамики ее изменения особенно значима для мобильных сетей.
- **Широкая известность NS2, как эталонного средства моделирования,** свидетельствующая о высокой степени достоверности результатов.
- **Свободное распространение пакета.**

  Симулятор NS2 относится к свободно распространяемым программным средствам (freeware), На сайте разработчиков [2] доступно несколько версий симулятора, ориентированных на различные операционные системы и аппаратные платформы. Наиболее полной является версия, ориентированная на семейство ОС UNIX, которая и рассматривается в данном пособии.

Как правило, совместно с симулятором NS2 используются дополнительные программные средства графического отображения результатов моделирования. В первую очередь, к ним относятся утилита сетевой анимации **nam** (network animator) и утилита построения графиков **X-graph**. Кроме того, вместе с симулятором распространяются средства отладки моделей, конвертации выходных данных, генераторы топологий сетей и сценариев моделирования, а также ряд других пакетов.

Состав симулятора постоянно обновляется, в данном пособии рассмотрены лишь основные его возможности. Более полная информация об установке, составе и настройке отдельных пакетов содержится в [2].

1.2. Общие принципы моделирования

В симуляторе исходные данные задаются с помощью программы-сценария - скрипта (Рис.1). Скрипт представляет собой текстовый файл, написанный на командном языке симулятора. При запуске симулятора, скрипт интерпретируется модулем событийного моделирования с использованием внутреннего планировщика событий. Никакой компиляции скрипта не требуется.

Язык симулятора предоставляет средства для простого описания топологии сети, конфигурации источников и приемников трафика, параметров линий связи (пропускной способности, задержки, вероятности потерь пакетов), учета специфики прикладных процессов и ряда других особенностей реальных коммуникационных систем.

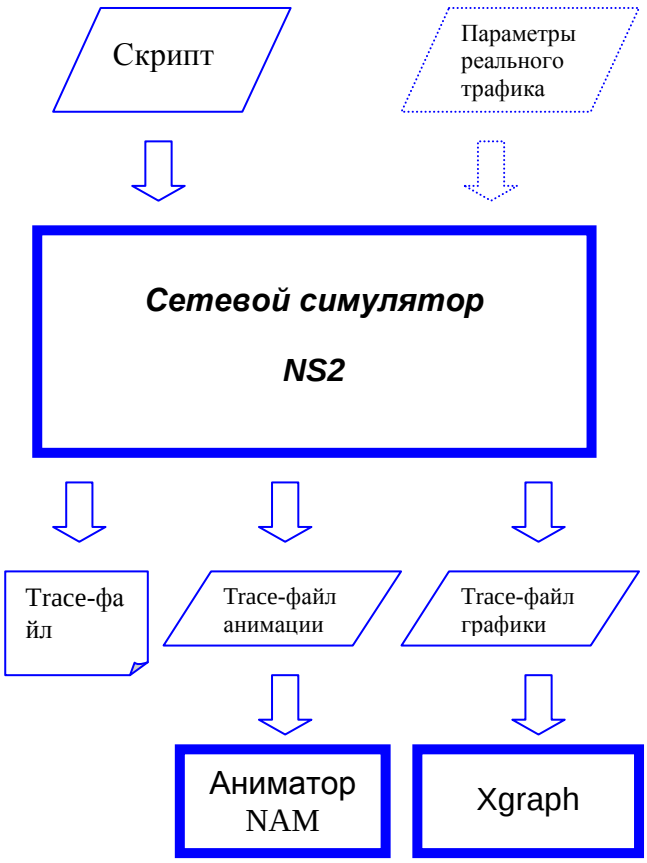


Рис.1 Исходные данные и результаты моделирования

Предусмотрена возможность задания потоков на основе параметров реального трафика (TCPdump). При моделировании симулятор позволяет легко изменять характеристики сети и потоков данных, варьировать параметры протоколов и размеры

буферных устройств, осуществлять мониторинг отправленных пакетов, сбор статистических данных и т.д.

В результате моделирования формируются регистрационные (trace-) файлы, содержащие информацию о динамике трафика, состоянии отдельных соединений и объектов сети, степени загрузки различных сетевых ресурсов, количестве переданных и потерянных пакетов и т.п. В регистрационных файлах могут также фиксироваться текущие значения внутренних переменных протоколов (sshtresh, congestion window, и т.д.).

Таким образом, в общем виде, процесс моделирования включает:

1. Создание скрипта в соответствии с целью моделирования.
2. Событийное моделирование (запуск симулятора) в соответствии с созданным скриптом.
3. Анализ и обработку результатов моделирования.

Предмет моделирования, вид и характер результатов, допущения о влиянии внешних факторов на работу сети полностью (явно или по умолчанию) определяются скриптом. Использование дополнительных программных средств, с целью обработки результатов моделирования, как правило, также требует наличия в скрипте соответствующих инструкций.

Следует отметить, что Network Simulator является программной моделью, управляемой событиями на пакетном уровне, – поэтому в интервале между различными событиями (каковыми являются, например, приход пакета или его отправка из буфера) ничего не происходит. Передача информации в симуляторе “осуществляется” только целыми пакетами, и нельзя получить сведения о том, какая часть пакета передана к заданному моменту времени. По умолчанию, пакеты направляются по наикратчайшему пути, метрикой маршрута считается число промежуточных узлов между приемником и источником. В симуляторе предусмотрена возможность изменения механизма маршрутизации пакетов.

1.2.1. Создание скрипта

Скрипт пишется на командном языке симулятора, являющемся своеобразным расширением языка OTcL - объектно-ориентированной версии языка TcL (Tool Command Language).

Данная версия симулятора также поддерживает использование скриптов, написанных на языке TcL, являвшимся основным командным языком в предыдущих версиях симулятора. Список основных объектов симулятора и описание относящихся к ним методов и свойств приведены в разделе 1.3.

Для написания скрипта можно использовать любой текстовый редактор. В операционной среде UNIX это могут быть редакторы: vi, pico, midnight commander (F4) и др.

1.2.2 Структура скрипта

В начале скрипта необходимо создать новый объект класса **Simulator** - в этом классе содержатся все методы, необходимые для дальнейшего описания модели. Это можно сделать командой:

```
set ns [new Simulator]
```

Для создания и уничтожения объектов используются методы **new** и **delete**. Следует отметить, что при создании скрипта, в ряде случаев, используется смесь языка TCL с внутренним языком симулятора (в данном примере команды внутри скобок представляют собой внутренние команды класса Simulator, а команда set ns - является командой языка TCL, используемая для присвоения имени нового объекта типа Simulator переменной 'ns').

Как правило, последняя строка скрипта, запускает симулятор

```
$ns run
```

.

Моделирование начинается по команде *run* и продолжается до тех пор, пока не будут обработаны все заданные события или до команды *stop.*

Выход из процесса моделирования осуществляется стандартной командой *exit.*

В общем случае, скрипт должен включать описания:

- топологии сети, включающей перечень узлов и характеристики каналов связи,
- характеристик потоков данных и используемых протоколов
- хронологии внешних событий (открытие и закрытие сеансов связи, отказы, моменты начала и конца моделирования и т.п.)
- формата вывода результатов моделирования.

В зависимости от целей моделирования, скрипт может содержать описания протоколов, используемых на различных уровнях взаимодействия в узлах сети, характеристики потоков отказов, модели помех и т.д. В текст скрипта можно включать комментарии, которые указываются в виде:

<комментарий>.

Причем, с символа # должна начинаться каждая строка комментария.

Корректность (непротиворечивость) скрипта, не проверяется симулятором и возлагается на

исследователя. Поэтому, при написании скрипта, следует четко определить цели моделирования и приемлемые допущения. Убедиться в адекватности сформулированных описаний топологии, протоколов, временных интервалов здравому смыслу.

Топология сети конструируется с помощью двух основных функциональных блоков: узлов и каналов связи. У класса Simulator имеются методы для создания и конфигурирования каждого из этих объектов. Узлы создаются при помощи метода **node**, и каждому узлу автоматически присваивается уникальный адрес. Каналы между узлами могут быть заданы с помощью симплексных (simplex-link) или дуплексных (duplex-link) методов **-link** , в результате чего моделируются однонаправленные или двунаправленные каналы связи. При описании каналов связи указывается пропускная способность канала, задержка распространения и механизм обработки очереди.

Например, команда

$ns simplex-link node1 node2 BWkb DELAYms type

– создает новый однонаправленный канал связи между узлами node1 и node2 с пропускной способностью, равной BW (в Кбитах в секунду), и с задержкой распространения, равной DELAY (в миллисекундах). Оба узла уже должны быть созданы при помощи метода node. Механизм обработки очереди в соединении определяется полем type, которое может принимать значения:
DropTail, FQ, SFQ, DRR, RED, CBQ, CBQ/WRR,
соответствующие механизмам типа FIFO, случайному выбору и др. :.

В соответствии с целью моделирования, каналы связи могут дополнительно характеризоваться заданным уровнем потерь.

Подробное описание механизмов обработки очередей и используемых моделей потерь, приведено в разделе 1.3.

При описании топологии локальных сетей в симуляторе используется метод **make-lan** и специальный объект **LanNode,** моделирующий использование общего ресурса. Особенности моделирования локальных сетей подробно рассмотрены в разделе 4.1.

Характеристики потоков данных и протоколов описываются с помощью объектов типа **Agent**.(Агент). Агенты представляют собой объекты, активно управляющие процессами в сетях. Они могут рассматриваться, как модели процессов и/или протоколов прикладного, транспортного или сетевого уровня, функционирующих в узлах. Примерами

агентов являются: источники трафика различного типа, приемники (**sinks**), маршрутизирующие модули, модули различных протоколов транспортного уровня.

Агенты создаются с помощью методов **Agent/*type*** , где ***type*** определяет вид конкретного агента общего класса **Agent.**

Например, TCP агент может быть создан с помощью следующей команды:

```
set tcp [ new Agent/TCP ]
```

После создания, агенты закрепляются за узлами при помощи метода **attach-agent.** Каждому агенту присваивается уникальный адрес порта для заданного узла (аналогично портам tcp и udp). Некоторые агенты могут иметь закрепленные за ними источники трафика, тогда как другие способны генерировать свои специфические потоки данных. Например, можно прикрепить “ftp” или “telnet” источники к “tcp”-агенту, тогда как “CBR”- агенты (constant bit rate - источник трафика с постоянной скоростью) генерируют регулярные потоки данных. Источники прикрепляются к агентам с помощью методов **attach-source** и **attach-traffic**.

Большинство объектов имеет ряд конфигурационных параметров, которые определяются в начале моделирования явно или по умолчанию и, в ряде случаев, могут быть модифицированы в процессе моделирования.

Например, размер окна приемника в TCP сессии может быть изменен следующим образом:

```
$tcp set window_ 25
```

Значения конфигурационных параметров всех классов объектов, устанавливаемые по умолчанию, могут быть легко переустановлены. Например, в результате выполнения команды

```
Agent/TCP set window_ 30
```

все создаваемые TCP агенты будут по умолчанию иметь размер окна равный 30 пакетов.
При описании протоколов следует внимательно следить за соответствием взаимодействующих агентов (источник – приемник) одному типу протокола.

Хронология событий, т.е. указание моментов начала и конца моделирования,

активизации и останова тех или иных агентов или процедур, осуществляется с помощью команды **at**. Например, команда

```
$ns at 0.5 “$ftp1 start”
```

запускает агент приложения ftp1 через 0,5 сек после начала моделирования.

Подобная простая привязка ко времени делает процесс моделирования достаточно гибким и удобным: в определенные моменты времени могут быть задействованы или отключены те или иные источники данных, запись статистики, разрыв, либо восстановление соединений, реконфигурация топологии и т.д.

Формат вывода определяет, что именно является предметом исследования, и каким образом предполагается обрабатывать результаты моделирования. В скрипте можно задать направление результатов на стандартный поток вывода **stdout**., сохранение информации о состоянии или динамике какого-либо фрагмента сети в регистрационном файле (tracefile) симулятора, запись результатов в соответствующие выходные файлы для последующей обработки утилитой анимации **nam** или утилитой построения графиков **X-graph.**

При сохранении результатов в файле, в общем случае, в скрипте необходимо определить файл для вывода результатов моделирования и указать объект сети, информация из которого будет поступать в данный файл:

Например,

```
set f [open out.tr w]
$ns trace-queue $n0 $n1 $f
```

Первой командой открывается файл регистрации out.tr для записи и ему присваивается управляющая переменная f. Следующая команда определяет, что для наблюдения будет использоваться очередь в канале между узлами n0 и n1, а информация будет направляться в файл f.

При использовании утилит **nam** и/или **X-graph** результаты моделирования должны быть представлены в регистрационных файлах соответствующих форматов. Причем, для вывода результатов в формате утилиты **nam** в симуляторе предусмотрена специальная команда. Утилита **nam** подробно рассматривается в разделе 2.

Утилита **X-graph** является универсальной утилитой построения графиков и требует явного преобразования формата trace-файла симулятора. Это преобразование может быть

выполнено в процессе моделирования (в этом случае оно должно быть предусмотрено в скрипте) или отдельно перед запуском утилиты **X-graph.** Более подробно использование утилиты **X-graph** рассмотрено в разделе 3.

1. 3. Основные объекты симулятора

Симулятор поддерживает большое разнообразие связанных объектов (Рис.3.1).

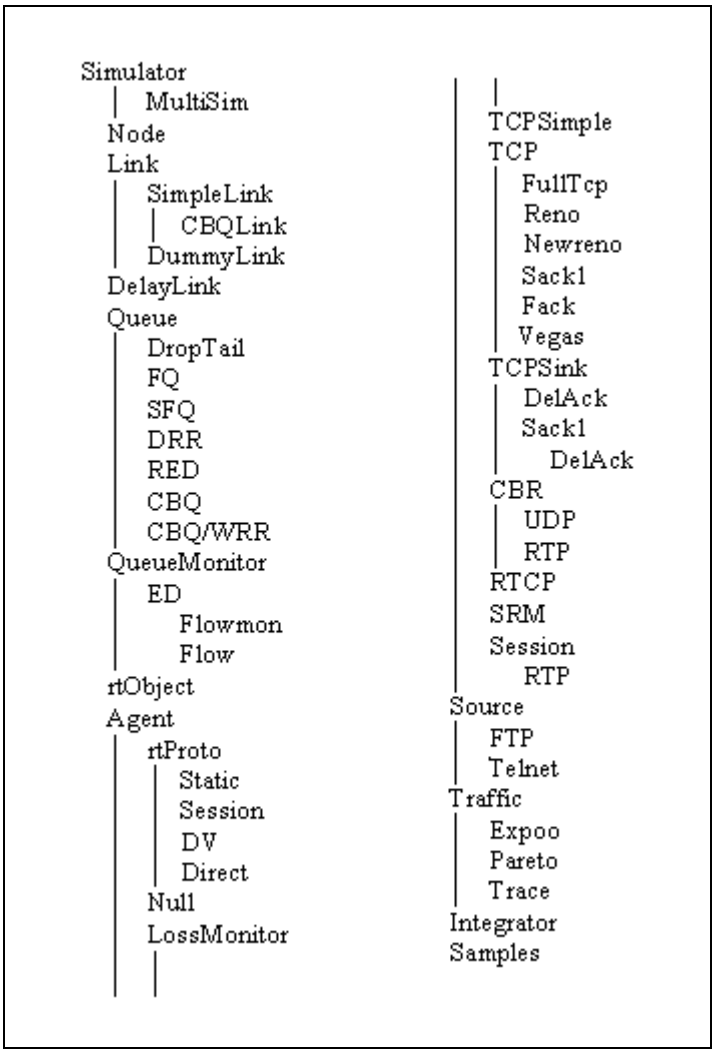


Рис. 3.1 Иерархия основных объектов симулятора

В этом разделе кратко описаны основные объекты (функциональные блоки) симулятора, такие, как узлы, каналы связи и агенты. Для каждого объекта приведены команды создания объектов данного класса, методы, относящиеся к объектам данного класса и необходимые конфигурационные параметры.

Объекты типа **Simulator**

Как уже упоминалось, первым шагом при моделировании является создание класса Simulator, который имеет методы конфигурации и запуска моделирования. При создании объекта типа Simulator происходит выполнение следующих действий:

- инициализируется модель пакетов, воспроизводящих пакетную передачу;
- создается планировщик событий;
- создается объект типа "null-агент", используемый в качестве приемника отброшенных и не регистрируемых пакетов.

Основные команды, относящиеся к объекту типа Simulator:

set ns [new Simulator] - создает объект типа Simulator и связывает его с переменной ns;

$ns now - команда возвращает текущее время моделирования;

$ns halt - приостановка планировщика (sheduler) симулятора;

$ns run - запуск планировщика симулятора;

$ns at *time event* - запуск операции *event* в момент времени *time*;

$ns cancel *event* - отменяет запланированное событие *event*, исключая его из списка операций планировщика;

Далее в этом разделе считается, что объект класса Simulator определен в переменной ns.

Объекты типа **Node** (Узел)

Основные команды, относящиеся к объектам типа Node:

$ns node - команда создает объект типа Node и возвращает ссылку на него.

$node id - возвращает идентификатор узла;

$node neighbors - возвращает список соседних узлов;7

$node attach *agent* - прикрепляет агент типа *agent* к узлу;

$node detach *agent* - отменяет прикрепление агента типа *agent* к узлу;

$node agent *port* - возвращает ссылку на агент, прикрепленный к порту *port* на данном узле. Возвращает пустую строку, если порт не используется.

$node reset - отменяет прикрепление всех агентов на данном узле и реинициализирует все переменные, связанные с агентами данного узла.

Объекты типа **Link** (связующее звено).

Основные команды, относящиеся к объектам типа Link:

$ns simplex-link *node1 node2 bw delay type* - создает новое однонаправленное соединение (звено связи) между узлами *node1* и *node2* с пропускной способностью

равной *bw* бит в секунду и задержкой распространения равной *delay* в секундах. Оба узла уже должны быть созданы при помощи метода *node*. Значения *bw* и *delay* по умолчанию 1.5 Мб/с и 100 мс соответственно. Значения по умолчанию могут быть изменены с помощью модификации соответствующих параметров объекта DelayLink. Поле *type* определяет механизм обработки очереди в соединении и может принимать значения: DropTail, FQ, SFQ, DRR, RED, CBQ, CBQ/WRR, описание которых приведено при рассмотрении объекта Queue .

$ns duplex-link *node1 node2 bw delay type* - создает новое двунаправленное соединение между узлами *node1* и *node2* с пропускной способностью равной *bw* бит в секунду и с задержкой распространения равной *delay* в секундах.

$ns link node1 node2 - возвращает ссылку на соединение между узлами node1 и node2. Эту команду удобно использовать при задании параметров соединения и методов мониторинга.

Методы, относящиеся к объектам типа Link:

$link trace-dynamics ns *fileID* - отслеживает динамику (события) для данного соединения и записывает информацию в файл с идентифицирующей переменной *fileID*. Поле ns - определяет класс Simulator (ns),

Методы управления разрывом/восстановлением линии связи

$ns rtmodel *model model-params node1* [*node2*] - восстанавливает/разрывает соединение между узлами *node1* и *node2* в соответствии с моделью *model*. Если определен только один узел *node1*, то действие будет применено ко всем линиям, связанным с данным узлом. Поле *model-params* содержит, в виде списка, все параметры, необходимые для определения модели. Параметры должны быть заключены в фигурные скобки. Поле *model* может принимать одно из следующих значений: **Deterministic**, **Exponential**, **Manual** или **Trace.**

В модели **Deterministic** параметры (*model-params)* имеют вид: [*start-time*] *up-interval down-interval* [*finish-time*]. Начиная с момента *start-time,* соединение восстанавливается на время *up-interval* и разрывается на время *down-interval*, периодически, до момента *finish-time*. Значения по умолчанию для *start-time, up-interval* и *down-interval,* соответственно, 0.5с, 2.0с, 1.0с. По умолчанию значение *finish-time* равно времени окончания моделирования.

При использовании модели **Exponential** параметры записываются в виде: *up-interval down-interval*. Времена восстановления и разрыва соединения выбираются

из экспоненциального распределения со средними значениями *up-interval* и *down-interval,* соответственно. Значения по умолчанию для средних *up-interval* и *down-interval* - 10.0с и 1.0с.

При использовании модели **Manual**, в качестве *model-params* применяется конструкция at *time op*, где *time* определяет время начала выполнения операции *op.* Значение поля *op* может быть *up* или *down*. Альтернативой данной модели может быть метод *rtmodel-at*, описанный ниже.

Модель **Trace** используется в случае, когда информация о событих, определяющих динамику узлов/соединений, читается из trace-файла. Аргумент *model-params* представляет собой ссылку на trace-файл, в котором находится информация о динамике. Формат trace-файла идентичен формату файла, генерируемого при мониторинге соединений/узлов (см. раздел Методы TRACE и MONITORING).

$ns rtmodel-delete *model-handle* - уничтожает модель, определенную полем *model-handle*;

$ns rtmodel-at *time op* node1 [node2] - используется для указания времени восстановления/разрыва линий связи между узлами node1 и node2. Если определен только один узел node1, то команда будет применена ко всем линиям, связанным с данным узлом. Поле *time* определяет время выполнения операции *op*, которая может принимать значения up или down, по отношению к определенному соединению.

Объекты маршрутизации **Static, Session, DV** позволяют создавать различные модели выбора маршрутов.

Команда задания конкретного объекта маршрутизации:

$ns rtproto *proto* node-list - определяет протокол (объект) маршрутизации *proto,* который будет использоваться на узлах, определенных в списке node-list. Протокол *proto* может быть одним из следующих типов: **Static, Session** или **DV.**

В объектах маршрутизации типа Static и Session используется алгоритм кратчайшего пути SPF (Shortest Path First). Протокол динамической маршрутизации DV использует алгоритм Беллмана-Форда (Bellman-Ford).

Используемый протокол маршрутизации можно задать с помощью процедуры rtproto{}, которая имеет ряд параметров: первый из которых определяет тип протокола, а остальные – узлы, для которых будут запущены протокольные модули маршрутизации. По умолчанию, протокол маршрутизации устанавливается единым

для всех узлов моделируемой сети. Если в скрипте не определено ни одной команды rtproto{}, то симулятор устанавливает протокол маршрутизации типа Static для всех узлов сети..

Объекты маршрутизации типа **Static** используются в симуляторе по умолчанию для вычисления маршрутов трафика. Процедура расчета маршрутов в сети запускается единственный раз в начале моделирования. Алгоритм вычисления использует цены (весовые коэффициенты) каналов связи. Если в процессе моделирования происходит изменения топологии сети, то некоторые источники и приемники трафика могут стать недоступны друг для друга.

Объекты маршрутизации типа **Session** по своей реализации аналогичны объектам типа Static, так как использует тот же алгоритм расчета маршрута. Отличием является то, что при использовании данного вида маршрутизации, кроме изначального расчета маршрутов, производится перерасчет маршрутов трафика при каждом изменении топологии в процессе моделирования. Это позволяет моделировать сам факт изменения маршрута прохождения пакетов одного соединения в конкретный момент времени. (Без моделирования процессов обмена маршрутизирующей информацией между узлами.)

Объекты маршрутизации типа **DV** позволяют более детально моделировать реальные процессы динамической маршрутизации и используют для расчета алгоритм Distance Vector. При использовании данного алгоритма, периодически, через заданный интервал времени, посылается информация об обновлении маршрутов топологии (по умолчанию, значение данного интервала (advertInterval) установлено равным 2 сек). Первое обновление информации отсылается агентом через 0.5 сек после начала моделирования. Кроме периодически обновляемой информации, каждый маршрутизирующий агент дополнительно отсылает информацию обновления маршрутов сети, при изменении таблиц пересылки данных на данном узле. Это происходит в случае изменения топологии или при получении агентом узла обновленной информации маршрутизации и перерасчете и установке новых маршрутов.

Команда вычисления маршрутов:

$ns compute-routes - вычисляет маршруты между всеми узлами в сети. Может

быть использована совместно с маршрутизацией типа static в случае, когда расчет маршрутов уже произведен, а состояние соединений изменилось, и маршруты необходимо обновить.

Объекты типа **Queue** (очередь) - основной класс объектов управления пакетами, при их движении по моделируемой сети. Данные объекты входят в состав линий связи.

$ns queue-limit node1 node2 *queue-limit* - устанавливает максимальное число пакетов, которое может находиться в очереди линии связи от узла node1 к узлу node2, равное *queue-limit*. Линия связи между узлами node1 и node2 уже должна быть создана.

Конфигурационные параметры:

*limit_* - размер очереди в пакетах, по-умолчанию 50 пакетов;

*blocked_* - режим блокировки. Принимает значения: true - если очередь блокирована (т.е. не способна посылать пакеты соседу по линии); иначе - false. По умолчанию установлено значение false.

Объекты типа **Drop-Tail** являются подклассом объектов типа Queue и используют простейший алгоритм обработки очереди - FIFO. Для данного подкласса не определены никакие методы, конфигурационные параметры или переменные состояния.

Объекты типа **FQ** являются подклассом объектов типа Queue и используют алгоритм обработки Fair Queuing.

Конфигурационные параметры

*secsPerByte_*

Объекты типа **SFQ** являются подклассом объектов типа Queue и используют алгоритм обработки Stochastic Fair Queuing. Для данного подкласса не определены никакие методы.

Конфигурационные параметры

*maxqueue_*

*buckets_* - число используемых областей памяти (буферов),

Объекты типа **DRR** являются подклассом объектов типа Queue и используют алгоритм "Deficit

round robin scheduling". Эти объекты обрабатывают на основе данного алгоритма пакеты различных потоков (т.е. пакеты с одинаковым идентификаторами узла и порта или только узла). В отличие от других объектов с обработкой нескольких очередей, объекты DRR используют единую область для буфера, которая делится между потоками.

Конфигурационные параметры

*buckets_* - число областей памяти (буферов), используемых для смешивания потоков;

*blimit_* - размер используемого буфера в байтах;

*quantum_* - число байт, которое может быть послано каждым потоком за отведенное ему время;

*mask_* - маска. При значении 1 - означает, что отдельный поток состоит из пакетов, имеющих одинаковые идентификаторы узла (и, возможно, различные идентификаторы порта), в противном случае (значение 0) - поток состоит из пакетов с одинаковыми идентификаторами и порта и узла.

Объекты типа **RED** являются подклассом объектов типа Queue и используют алгоритм "Random Early-Detection". Объект может быть сконфигурирован, как на отбрасывание, так и на "пометку" пакетов. Для данных объектов не определено никаких методов.

Конфигурационные параметры:

*bytes_* - установка в true означает "byte-mode" RED, где размер прибывающих пакетов влияет на вероятность отбрасывания ("отметки") пакетов.

*queue-in-bytes_* - установка в true показывает, что размер очереди измеряется в байтах, а не в пакетах. Установка этой опции, также приводит к автоматическому изменению параметров *thresh_* и *maxthresh_* с помощью *mean_pktsize_* (см. далее).

*thresh_* - минимальная граница для среднего размера очереди в пакетах.

*maxthresh_* - максимальная граница для среднего размера очереди в пакетах.

*mean_pktsize_* - оценка среднего размера пакета в байтах.

*q_weight_* - вес очереди, - используется для вычисления среднего размера очереди.

*wait_* - при установке в true устанавливается интервал между отброшенными пакетами.

*linterm_* - используется для расчета вероятности отбрасывания пакета,

которая, в этом случае, варьируется между 0 и "1/linterm".

*setbit_* - при установке в "true" пакеты не отбрасываются, а "помечаются" (в пакетах устанавливается бит, характеризующий перегруженность).

*drop-tail_* - при установке в "true", в случае переполнения очереди, вместо механизма randomdrop используется механизм drop-tail.

Объекты типа **CBQ** являются подклассом объектов типа Queue и используют механизмы обработки очереди, зависящие от класса трафика (Class-Based-Queueing)

Основные команды, связанные с объектами типа CBQ:

$cbq insert $*class* - добавляет класс трафика *class* в имеющуюся структуру типов трафика, соответствующую данному объекту *cbq*.

$cbq bind $*cbqclass* $*id1* [$*id2*] - устанавливает соответствие между пакетами с идентификатором потока $*id1* (или диапазона $*id1* - $*id2*) и классом трафика $*cbqclass*.

$cbq algorithm $*alg* - определяет внутренний алгоритм CBQ. $*alg* может быть установлено в одно из значений: "ancestor-only", "top-level" или "formal".

Объекты типа **CBQ/WRR** являются подклассом объектов CBQ, в которых используется метод "weighted round-robin scheduling" между классами одного приоритетного уровня. В CBQ объектах используется попакетное (packet-by-packet) управление (round-robin scheduling).

Конфигурационные параметры

*maxpkt_* - максимальный размер пакета в байтах. Значение этого параметра используется только CBQ/WRR-объектами для вычисления максимальной выделяемой пропускной способности.

Объекты типа **QUEUEMONITOR** используются для мониторинга получаемых, отправляемых и отбрасываемых пакетов и байтов. Они также поддерживают вычисление статистики (средний размер очереди и т.д.)

Основные команды:

$queuemonitor - сбрасывает все, описываемые далее, счетчики (прибывших, отправленных и отброшенных байт) в ноль. Также сбрасывает значения интеграторов и элементов задержки, если они определены.

$queuemonitor set-delay-samples *delaySamp_* - устанавливает Samples объект *delaySamp_* для записи статистики о задержках очереди. Объект Samples с управляющей переменной *delaySamp_* должен быть уже создан.

$queuemonitor get-bytes-integrator - возвращает состояние объекта типа Integrator, который может быть использован для вычисления интегрального значения размера очереди в байтах (см. раздел Объекты типа Integrator).

$queuemonitor get-pkts-integrator - возвращает состояние объекта типа Integrator, который может быть использован для вычисления интегрального значения размера очереди в пакетах (см. раздел Объекты типа Integrator).

$queuemonitor get-delay-samples - возвращает состояние объекта типа Samples delaySamp_ для записи статистики о задержках очереди (см. раздел Объекты типа Samples).

Переменные состояния

*size_* - текущий размер очереди в байтах;

*pkts_* - текущий размер очереди в пакетах;

*parrivals_* - общее число полученных пакетов;

*barrivals_* - общее число полученных байт;

*pdepartures_* - общее число отправленных пакетов (не отброшенных);

*bdepartures_* - общее число байт, содержащееся в отправленных пакетах (не отброшенных);

*pdrops_* - общее число отброшенных пакетов;

*bdrops_* - общее число отброшенных байт;

*bytesInt_* - объект типа Integrator, который вычисляет интегральное значение очереди в байтах. Параметр sum_ содержит интегральное значение размера очереди в байтах.

*pktsInt_* - объект типа Integrator, который вычисляет интегральное значение очереди в пакетах. Параметр sum_ содержит интегральное значение размера очереди в пакетах.

Объекты типа **Agent** (Агент) моделируют работу протоколов различного уровня в узлах сети. Основные команды:

new Agent/*type* - создает агент типа *type*, который может принимать значения:

Null - простейший приемник трафика;

LossMonitor – приемник с мониторингом потерянных пакетов;

TCP - TCP версии BSD Tahoe;

CBR - источник трафика с постоянной скоростью;

и т.д.

$ns attach-agent *node agent* - прикрепляет объект *agent* к узлу *node*. Объекты *node* и *agent* должны быть уже созданы.

$ns detach-agent *node agent* - отменяет закрепление объекта *agent* за узлом node.

$ns connect src dst - создает двунаправленное соединение между агентами src и dst.

Методы, относящиеся к объектам типа Agent:

$agent port - возвращает значение порта для агента;

$agent dst-addr - возвращает адрес узла назначения, с которым соединен данный агент;

$agent dst-port - возвращает значение порта узла назначения, с которым соединен данный агент;

$agent attach-source *type* – прикрепляет источник данных типа *type* к агенту (см. соответствующие методы агентов для информации о конфигурационных параметрах.). Возвращает ссылку на объект источник.

$agent attach-traffic *traffic-object* - прикрепляет генератор трафика *traffic-object* к агенту.

Возможные типы источников трафика:

Traffic/CBR –источник трафика с постоянной скоростью,

Traffic/Expoo - генератор трафика с экспоненциальным распределением интервалов On/Off.

Traffic/Pareto - генератор трафика с Pareto-распределением промежутков On/Off.

Traffic/Trace - генерирует трафик на основе trace-файла.

Соответствующие конфигурационные параметры описаны в разделе "Объекты типа TRAFFIC".

$agent connect *addr port* – устанавливает соединение агента agent с агентом, имеющим адрес *addr* и порт *port*. Два данных агента должны быть совместимы (например,. tcp-source - tcp-sink), в противном случае, результаты моделирования могут быть непредсказуемы.

Конфигурационные параметры

*dst_* - адрес агента назначения, с которым соединен данный агент. Обычно состоит из 32 бит, из которых 24 бита определяют идентификатор узла назначения, а оставшиеся 8 бит - номер порта.

Объекты типа **Null** - подкласс объектов агентов, являющихся приемниками трафика. Null объекты не имеют никаких методов, конфигурационных параметров и переменных состояния.

Объекты типа **LossMonitor** - подкласс объектов агентов, являющихся приемниками трафика, которые поддерживают сбор статистики о полученных данных, т.е. число полученных байт, число потерянных пакетов и т.д..

Параметры состояния

*nlost_* - число потерянных пакетов;

*npkts_* - число полученных пакетов;

*bytes_* - число полученных байт;

*lastPktTime_* - время получения последнего пакета;

*expected_* - ожидаемый порядковый номер (sequence number) следующего пакета.

Объекты типа **ErrorModel_** используются для моделирования каналов связи с ошибками передачи или потерями. В канале с ошибками пакеты могут быть отброшены целиком или у них может быть установлен так называемый флаг ошибки. Объект ErrorModel может быть вставлен в объекты типа SimpleLink. Поскольку объекты SimpleLink являются составными объектами, генераторы ошибок могут быть вставлены в различные позиции:

- до очереди соединения, (команда - $simplelink errormodule *args)*;
- после очереди, но до модуля "delay link" ($simplelink insert-linkloss *args)*;
- после модуля "delay link" ($simplelink install-error).

Объекты типа **TCP** представляют собой класс объектов агентов, моделирующих транспортный протокол TCP Tahoe.

Конфигурационные параметры

*window_* - верхняя граница окна приемника (Advertisment Window) TCP соединения;

*maxcwnd_* - верхняя граница окна переполнения TCP соединения. Для отмены ограничения устанавливается в 0 (значение по умолчанию).

*windowInit_* - начальное значение окна переполнения для медленного старта.

*windowOption_* - тип алгоритма, использующегося для управления окном переполнения.

*windowThresh_* - постоянная сглаживающего фильтра, используемого для вычисления awnd (см. далее). Применяется для исследования различных алгоритмов управления окном.

*overhead_* - диапазон случайно распределенной переменной, используемой для задержки каждого выходного пакета. Применяется только в версии tcp Tahoe, в tcp Reno не используется.

*ecn_* - указатель использования (true/false) механизма явного оповещения (explicit congestion notification) в дополнение к отбрасыванию пакетов, при насыщении.

*packetSize_* - размер пакетов источника;

*tcpTick_* - интервал таймера TCP, используемый для оценки времени RTT (round-trip time). По умолчанию установлена нестандартная величина 100ms.

*bugFix_* - указатель (true/false) запрета механизма быстрой ретрансмиссии при потере пакетов в одном окне данных. Установка в true запрещает быстрый повтор передачи нескольких пакетов, потерянных в одном окне данных.

*maxburst_* - максимальное число пакетов, которое может посылать источник в ответ на

одно полученное подтверждение. При отсутствии ограничения устанавливается в 0.

*slow_start_restart_* - признак использования (1/0) механизма медленного старта. Включено по умолчанию.

MWS (Maximum Window Size) – константа, определяющая максимальный размер окна (в пакетах), допустимый в симуляторе. По умолчанию MWS=1024 пакетам. Для Tahoe TCP параметр "window" представляет размер указанного окна получателя, которое должно быть меньше чем MWS-1. Для Reno TCP значение "window" должно быть меньше чем (MWS-1)/2.

Переменные состояния

*dupacks_* - число дублирующих подтверждений (ACK), полученных после прихода последнего недублирующего подтверждения

*seqno_* - наивысший последовательный номер сегмента (sequence number) для данных источника TCP;

*t_seqno_* - текущий последовательный номер сегмента (пакета) пакета;

*ack_* - наивысшее значение из полученных подтверждений ;

*cwnd_* - текущее значение окна переполнения;

*awnd_* - текущее значение окна переполнения при использовании усреднения. Используется для исследования различных алгоритмов увеличения окна.

*ssthresh_* - текущее значение порога медленного старта;

*rtt_* - оценка значения round-trip time;

*srtt_* - оценка сглаженного значения round-trip time;

*rttvar_* - оценка среднего отклонения значений round-trip time;

*backoff_* - экспоненциальная постоянная задержки для round-trip time;

Объекты типа **TCP/Reno** являются подклассом объектов TCP, моделирующим протокол Reno TCP. Для этих объектов не определены никакие дополнительные методы, конфигурационные параметры и переменные.

Объекты **TCP/NewReno** являются подклассом объектов TCP, моделирующим модифицированную версию протокола BSD Reno TCP.

Конфигурационные параметры

*newreno_changes_* - указатель версии протокола . Установка в 0 соответствует основной версии NewReno, установка в 1 приводит к использованию дополнительных алгоритмов NewReno

Объекты типа **TCP/Sack1** - подкласс объектов TCP, моделирующий протокол BSD Reno TCP с селективным подтверждением. Объекты Sack1 наследуют все функциональные особенности TCP объектов. Для этих объектов не определено никаких дополнительных методов, конфигурационных параметров и переменных.

Объекты типа **TCP/Fack** - подкласс объектов TCP, моделирующий протокол BSD Reno TCP с механизмом Forward Acknowledgement Congestion Control.

Объекты типа **TCP/Vegas** - подкласс объектов TCP, моделирующий протокол Vegas TCP.

Объекты типа **TCP/FullTcp** являются подклассом объектов TCP, более детально моделирующим существующие реализации протокола TCP. Отличия от других агентов TCP заключаются в следующем:

- Моделируется обмен пакетами установления и завершение соединения (SYN/FIN).
- Поддерживается двунаправленная передача данных.
- Последовательная нумерация (sequence number) относится к байтам, а не к пакетам.

Конфигурационные параметры

*segsperack_* - число сегментов, на которое генерируется одно подтверждение;

*segsize_* - размер сегмента;

*tcprexmtthresh_* - число дублирующих подтверждений, после получения которых осуществляется переход в режим быстрого повтора передачи;

*iss_* - начальный последовательный номер первого байта передаваемых данных;

*nodelay_*- отключение (false) алгоритма Найджела (Nagle);

*data_on_syn_* - разрешение/запрет (True/False) передачи данных совместно с пакетом SYN;

*dupseg_fix_* - исключение быстрого восстановления при получении дублирующих подтверждений;

*dupack_reset_* - сброс счетчика дублирующих подтверждений при получении сегментов нулевой длины, содержащих дублирующие подтверждения;

Объекты типа **TCPSink** представляют собой подкласс объектов агентов моделирующий протокол TCP на стороне приемника. Отметим, что объекты TCP (кроме FullTcp) моделируют только однонаправленные TCP соединения, в которых TCP-источник посылает пакеты данных,

а приемник – только подтверждения (ACK пакеты). Для этих объектов не определено никаких методов и переменных.

Конфигурационные параметры

*packetSize_* - размер используемых пакетов подтверждений в байтах;

*maxSackBlocks_* - максимальное число блоков данных, которое может быть подтверждено в опции SACK. Этот параметр используется только подклассом объектов TCPSink/Sack1. Эта величина не может быть увеличена для TCPSink объекта после того как объект создан. (После создания TCPSink объекта величина может быть уменьшена, но не увеличена).

Объекты типа **TCPSink/DelAck** являются подклассом объектов TCPSink, моделирующим TCP-приемник с механизмом задержанных подтверждений. Они наследуют функциональность TCPSink объектов.

Конфигурационные параметры

*interval_* - временная задержка перед генерацией подтверждения на отдельный пакет. Если следующий пакет прибывает до истечения данного времени, то подтверждение генерируется немедленно.

Объекты типа **TCPSink/Sack1** – подкласс объектов TCPSink, моделирующий TCP-приемник с селективным подтверждением пакетов. Для этих объектов не определено никаких дополнительных методов, конфигурационных параметров и переменных.

Объекты типа **TCPSink/Sack1/DelAck** - подкласс объектов TCPSink/Sack1, моделирующий TCP-приемник с задержанным селективным подтверждением пакетов.

Конфигурационные параметры

*interval_* - временная задержка перед генерацией подтверждения на отдельный пакет. Если другой пакет прибывает до истечения данного времени, то подтверждение генерируется немедленно.

Объекты типа **CBR** предназначены для моделирования источника данных, генерируемых непрерывно с постоянной битовой скоростью.

Основные команды:

$cbr start - команда начала генерации данных;

$cbr stop – команда прекращения генерации данных;

Конфигурационные параметры

*interval_* - задержка между генерацией пакетов;

*packetSize_* - размер пакетов источника в байтах;

*random_* - параметр определяет, присутствует ли случайный шум в процессе генерации пакетов. Если значение *random_* равно 0, то время между генерацией пакетов определяется параметром *interval_*, в противном случае, временной промежуток выбирается случайным образом из интервала [0.5·*interval_*, 1.5·*interval_*].

Объекты типа **Source** моделируют источники потоков данных, соответствующие основным приложениям стека TCP/IP.

.

Объекты типа **Source/FTP**

$ftp start - команда источнику Source/FTP сгенерировать *maxpkts_* пакетов;

$ftp produce *n* - команда источнику сгенерировать *n* пакетов;

$ftp stop - команда остановки пересылки данных;

$ftp attach *agent* - прикрепляет Source/FTP объект к агенту *agent*;

$ftp producemore *count* - команда источнику Source/FTP сгенерировать дополнительно *count* пакетов;

Конфигурационные параметры

maxpkts - максимальное число пакетов генерируемых источником. Моделирует размер передаваемого файла.

Объекты типа **Source/Telnet** используются для генерации отдельных пакетов с заданными интервалами. Если параметр interval_ не равен нулю, то времена между генерацией пакетов выбираются из экспоненциального распределения со средним interval_. Если параметр interval_ имеет значение ноль, то времена между пакетами выбираются с использованием распределения "tcplib" telnet.

Основные команды:

$telnet start - запуск источника;

$telnet stop - остановка источника;

$telnet attach *agent* - прикрепление объекта Source/Telnet к агенту *agent*;

Конфигурационные параметры:

*interval_* - среднее время между генерируемыми пакетами в секундах.

Объекты типа **Traffic** моделируют потоки данных с заданными характеристиками. Объекты типа Traffic создаются методами Traffic/*type*, где *type* определяет тип генератора потока и принимает значения: CBR, Expoo, Pareto или Trace.

Traffic/CBR –источник трафика с постоянной скоростью,

Traffic/Expoo - генератор трафика с экспоненциальным распределением промежутков On/Off.

Traffic/Pareto - генератор трафика с Pareto -распределением промежутков On/Off.

Traffic/Trace - генерирует трафик на основе trace файла.

Объекты **Traffic/Expoo** генерируют прерывистый “On/Off” трафик. В течение "On"-периодов пакеты генерируются с постоянной битовой скоростью. Во время "Off"- периодов трафик не генерируется. Интервалы "On" и "Off" являются случайными величинами, имеющими экспоненциальное распределение.

Конфигурационные параметры

packet-size - размер пакетов в байтах;

burst-time – среднее значение интервала генерации в секундах ("on"- период);

idle-time – среднее значение "off"-периода;

rate – скорость передачи (бит в секунду).

Объекты **Traffic/Pareto** также генерируют On/Off трафик. Временные интервалы "On" и "Off" являются случайными величинами, распределенными по закону Pareto.

Конфигурационные параметры

packet-size - размер пакетов в байтах;

burst-time – среднее значение интервала генерации в секундах ("on"- период);

idle-time – среднее значение "off"-периода;

rate – скорость передачи (бит в секунду).

shape - параметр распределения Парето.

Объекты **Traffic/Trace** генерируют трафик на основе файла данных. Это позволяет использовать характеристики реальных потоков данных.

Основные команды:

$trace attach-tracefile *tfile* - прикрепляет Tracefile-объект *tfile* к trace-объекту. Tracefile объект определяет файл, из которого будут читаться данные трафика (см. секцию TRACEFILE

объекты). К одному Tracefile объекту может быть прикреплено несколько Traffic/Trace объектов. Для каждого Traffic/Trace объекта выбирается случайное стартовое место в Tracefile.

Конфигурационные параметры для данного объекта не определены.

Объекты **Tracefile** используются для определения файла, на основе которого будет генерироваться трафик (см. секцию Traffic/Trace объекты).

Основные команды:

Объект типа Tracefile создается командой [new Tracefile]

$tracefile filename *fileinput* - устанавливает имя файла *fileinput,* из которого будут читаться данные для объекта Tracefile.

Конфигурационные параметры для данного объекта не определены.

Регистрационный файл *fileinput* может состоять из любого числа записей фиксированной длины. Каждая запись состоит из двух 32-х битных полей. Первое показывает время до момента генерации следующего пакета в микросекундах. Второе определяет длину следующего пакета в байтах.

Регистрация и обработка результатов моделирования

Для регистрации и обработки результатов моделирования в симуляторе предусмотрены следующие методы:

$ns create-trace *type fileID node1 node2* - создает регистрационный объект типа *type* для мониторинга очереди между узлами *node1* и *node2*. Тип *type* может принимать значения: Enque, Deque или Drop. Enque отслеживает прибывающие в очередь пакеты. Deque - отправляемые пакеты, а Drop – отброшенные. Переменная *fileID* является управляющей переменной открытого файла данных. Соответствующий файл должен быть открыт для записи.

$ns drop-trace *node1 node2 trace* - удаляет регистрационный объект с управляющей переменной *trace*, прикрепленный к соединению между узлами *node1* и *node2*

$ns trace-queue *node1 node2 fileID* - создает регистрационный объект, осуществляющий все типы (Enque, Deque и Drop) мониторинга очереди между узлами *node1* и *node2*..

$ns trace-all *fileID* - создает регистрационный объект, осуществляющий мониторинг всех событий (изменение состояния очередей, узлов и каналов) в моделируемой сети.

$ns monitor-queue *node1 node2* – создает объект типа QueueMonitor, позволяющий получать данные о состоянии очереди между узлами *node1* и *node2*. (входящие, выходящие, отброшенные пакеты и их средние значения).

$ns flush-trace – переносит данные из буферов в соответствующий регистрационный (trace) файл.

$link trace-dynamics *ns fileID* - создает регистрационный объект, осуществляющий мониторинг состояния (разрыв, восстановление и т.п.) данного соединения и записывающий данные в файл с управляющей переменной *fileID*. При этом, *ns* - переменная объекта типа Simulator или MultiSim.

Симулятор включает также ряд объектов, реализующих математическую обработку результатов моделирования. К ним относятся объекты типа **Integrator, Samples** и другие

1.4. Формат регистрационного файла симулятора

Регистрационный файл симулятора является текстовым файлом и содержит последовательность записей, соответствующих событиям в моделируемой сети. Все записи в последовательности упорядочены по времени наступления события. В зависимости от типа события, записи могут иметь различный формат.

Записи, характеризующие динамику пакетов, имеют формат:

<code> <time> <hsrc> <hdst> <type> <size> <flags> <flowID><src.sport> <dst.dport> <seq> <pktID>,

где

<code> := код события [d|+|-|r] d=drop +=enque -=deque r=receive;

<time> := текущее время моделирования в секундах;

<hsrc> := адрес первого узла соединения;

<hdst> := адрес второго узла соединения;

<type> := tcp|telnet|cbr|ack и т.д.

<size> := размер пакета в байтах;

<flags> := [CP] C=congestion (перегрузка), P=priority (приоритет);

<flowID> := поле идентификатора потока;

<src.sport> := адрес источника (src=node, sport=agent);

<dst.dport> := адрес назначения (dst=node, dport=agent);

<seq> := последовательный номер (sequence number) пакета;

<pktID> := уникальный идентификатор каждого пакета;

Записи, характеризующие состояние очереди.

Для соединений, использующих механизм обработки очереди типа RED, присутствуют дополнительные записи, имеющие формат:

<code> <time> <value>,

где

<code> := код параметра: [Q|a|p] Q=размер очереди, a=средний размер очереди, p=вероятность отброса пакетов;

<time> := текущее время моделирования в секундах;

<value> := значение параметра;

Записи, характеризующие состояние каналов связи, имеют формат:

<code> <time> <state> <src> <dst>,

где

<code> := код события [v];

<time> := текущее время моделирования в секундах;

<state> := [link-up (восстановление соединения) | link-down (разрыв)];

<src>, <dst> := адреса узлов определяющих канал связи.

Фрагмент регистрационного файла симулятора представлен на рис.3.

```
d 0 0 1 cbr 210 ------- 0 0.0 1.0 0 0
d 0.00375 0 1 cbr 210 ------- 0 0.0 1.0 1 1
+ 0.0075 0 1 cbr 210 ------- 0 0.0 1.0 2 2
- 0.0075 0 1 cbr 210 ------- 0 0.0 1.0 2 2
+ 0.01125 0 1 cbr 210 ------- 0 0.0 1.0 3 3
- 0.01125 0 1 cbr 210 ------- 0 0.0 1.0 3 3
+ 0.015 0 1 cbr 210 ------- 0 0.0 1.0 4 4
- 0.015 0 1 cbr 210 ------- 0 0.0 1.0 4 4
+ 0.01875 0 1 cbr 210 ------- 0 0.0 1.0 5 5
- 0.01875 0 1 cbr 210 ------- 0 0.0 1.0 5 5
d 0.0225 0 1 cbr 210 ------- 0 0.0 1.0 6 6
+ 0.02625 0 1 cbr 210 ------- 0 0.0 1.0 7 7
- 0.02625 0 1 cbr 210 ------- 0 0.0 1.0 7 7
+ 0.03 0 1 cbr 210 ------- 0 0.0 1.0 8 8
- 0.03 0 1 cbr 210 ------- 0 0.0 1.0 8 8
+ 0.03375 0 1 cbr 210 ------- 0 0.0 1.0 9 9
- 0.03375 0 1 cbr 210 ------- 0 0.0 1.0 9 9
+ 0.0375 0 1 cbr 210 ------- 0 0.0 1.0 10 10
- 0.0375 0 1 cbr 210 ------- 0 0.0 1.0 10 10
+ 0.04125 0 1 cbr 210 ------- 0 0.0 1.0 11 11
- 0.04125 0 1 cbr 210 ------- 0 0.0 1.0 11 11
+ 0.045 0 1 cbr 210 ------- 0 0.0 1.0 12 12
- 0.045 0 1 cbr 210 ------- 0 0.0 1.0 12 12
d 0.04875 0 1 cbr 210 ------- 0 0.0 1.0 13 13
d 0.0525 0 1 cbr 210 ------- 0 0.0 1.0 14 14
+ 0.05625 0 1 cbr 210 ------- 0 0.0 1.0 15 15
- 0.05625 0 1 cbr 210 ------- 0 0.0 1.0 15 15
r 0.05862 0 1 cbr 210 ------- 0 0.0 1.0 2 2
+ 0.06 0 1 cbr 210 ------- 0 0.0 1.0 16 16
- 0.06 0 1 cbr 210 ------- 0 0.0 1.0 16 16
r 0.06237 0 1 cbr 210 ------- 0 0.0 1.0 3 3
d 0.06375 0 1 cbr 210 ------- 0 0.0 1.0 17 17
r 0.06612 0 1 cbr 210 ------- 0 0.0 1.0 4 4
```

Рис. 3 Фрагмент регистрационного файла симулятора

2. Утилита анимации nam

2.1. Назначение

Утилита **nam** (Network Animator) предназначена для графического отображения результатов моделирования в симуляторе **ns2**. Аниматор обеспечивает графическое отображение топологии моделируемой сети, анимацию процессов на уровне пакетов, а также различные средства инспекции данных. В принципе, исходные данные для аниматора могут быть получены как в результате работы симулятора ns2, так и в результате исследований трафика реальной сети (например, в результате применения утилиты tcpdump). В последнем случае требуется дополнительно описать топологию сети.

2.2 Параметры запуска

Первым шагом, при использовании утилиты nam, является создание исходного trace-файла данных. Trace-файл должен содержать информацию о топологии системы, т.е. об узлах, линиях связи и т.д., а также собственно информацию о динамике пакетов данных. Детальный формат файла описан в разделе 'Формат Trace файла'. При использовании nam совместно с симулятором ns2, в результате моделирования сразу создается регистрационный файл в требуемом формате.

После того, как исходный файл данных создан, можно приступать к запуску анимации в nam. После запуска, nam считывает информацию из trace-файла, создает топологию, отображает ее в своем окне, при необходимости, соответствующим образом ее ориентирует, и, в выбранном масштабе времени, отображает движение пакетов в сети. Пользовательский интерфейс аниматора позволяет контролировать многие аспекты анимации. Эти возможности аниматора описаны в разделе 'Пользовательский Интерфейс'.

При вызове аниматора указывается исходный trace-файл (tracefile) и, в общем случае, ряд параметров:

nam [-**g** geometry] [-**t** graphInput] [-**i** inteval] [-**p** peerName] [-N appName]
[-**c** cashSize] [-**f** configfile] [-**S**] tracefile ,

где: tracefile – имя исходного файла данных

-g - определяет геометрию отображения при запуске.

-t - определяет использование утилиты nam совместно с утилитой tkgraph. Также определяет входной файл для tkgraph.

-i - определяет частоту обновления экрана в миллисекундах. Скорость обновления по умолчанию 50 миллисекунд (20 кадров в секунду).

-N - определяет имя приложения для данного запускаемого процесса nam.

-p - определяет имя приложения дополнительного процесса nam, работа которого может быть синхронизирована с работой текущего процесса nam.

Для этого может быть использована следующая последовательность команд;

nam -N <name#1> <trace file name #1> - запуск первого процесса nam (подчиненного);

nam -N <name#2> <trace file name #2> - запуск второго процесса nam (главного);

После этого, управление анимацией (воспроизведение, остановка, обратное воспроизведение и т.д.) двух процессов будет синхронизировано..

-c - определяет максимальный размер кэш-памяти, используемой для хранения 'активных' объектов при обратном воспроизведении.

-f - задает имя файла инициализации, загружаемого при запуске. В этом файле пользователь может определить функции, вызываемые из trace-файла. Примеры таких функций - 'link-up' и 'link-down' для моделирования сетей с отказами.

-S задает дополнительную синхронизацию графического вывода с X-Windows в системах Unix.

2.3. Пользовательский интерфейс

Пользовательский интерфейс аниматора (рис 2.1) включает панель меню, панель управления, основное окно аниматора, панель автоориентации графа сети и окно комментариев.

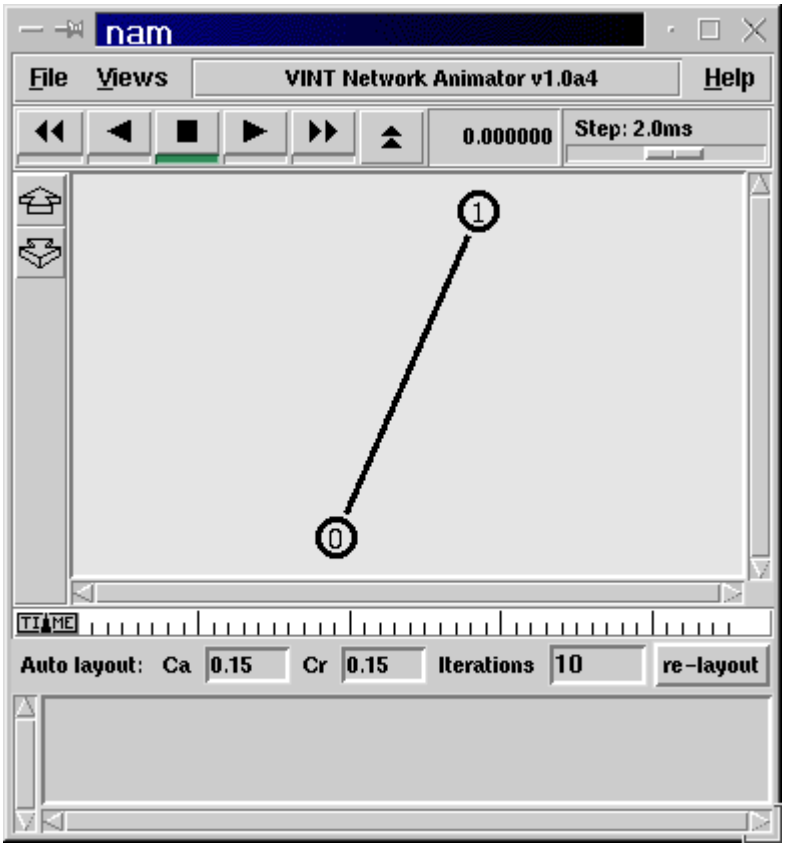


Рис. 2.1.

Панель меню включает следующие наборы возможных операций:

Меню 'File'

Save layout – сохранение текущей топологии в файле 'savefile';

Print layout - печать текущей топологии;

Record Animation - запись процесса анимации;

AutoFastForward – автоматическая анимации в ускоренном режиме;

Quit - выход из аниматора;

Меню 'Views'

New view - открытие нового окна аниматора.

Edit view - открытие нового окна аниматора для редактирования;

Show monitors – отображение информации об объектах мониторинга;

Show autolayout – отображение элементов управления автоориентацией графа сети.

Show annotation – отображение комментария о моделируемых событиях

Меню 'Analysis'

Active Sessions - открытие окна со списком активных на данный момент сессий;

Legend … - открытие окна с описанием условных обозначений;

Справа от указанных меню находится панель с указанием имени используемого trace файла и меню 'Help', содержащее справочную информацию об аниматоре.

Для вызова меню могут использоваться клавиши быстрого доступа:

Alt+'f' - открытие меню 'File', Alt+'v' - меню 'Views' и Alt+'a' - меню 'Analysis'.

Под панелью меню находится панель управления процессом анимации, включающая клавиши:

'<<' - Перемотка назад

'<' - Воспроизведение анимации в обратном направлении

□ - Остановка (пауза).

'>' - Воспроизведение анимации.

'>>' - Перемотка вперед.

'^' - Выход.

Здесь же находятся индикатор текущего времени анимации (время определенное в trace файле) и бегунок управления скоростью обновления экрана.

Под панелью управления располагается основное окно, в котором отображается моделируемая сеть. Слева от него находится панель с двумя кнопками, позволяющими масштабировать изображение. Если щелкнуть левой кнопкой мыши по любому объекту в главном окне, то появится всплывающее информационное окно. Для объектов типа пакеты и агенты в этом окне будет присутствовать кнопка 'Monitor', с помощью которой можно добавить панель мониторинга для этих объектов. Она появится в нижней части окна nam. Для объектов типа линия связи, информационное окно будет содержать кнопку 'Graph', при нажатии на которую будет вызвано еще одно всплывающее окно, в котором пользователь может выбрать, что будет отображаться на графике для данной линии = интенсивность потока пакетов, либо график потерь. Для дуплексного соединения можно выбрать для наблюдения любое направление.

2.4. Формат Trace-файла утилиты nam

Исходный файл (trace-файл) данных аниматора является текстовым файлом и содержит последовательность записей, соответствующих событиям в моделируемой сети. Все записи в последовательности упорядочены по времени наступления события.

В начале файла могут присутствовать записи без явного указания времени (поле –t содержит звездочку *). Эти записи соответствуют “установочным” событиям (указание версии аниматора, создание топологии сети, исходное задание цвета и т.п.).

Записи могут иметь различный формат, в зависимости от объекта, к которому они относятся. Основными объектами аниматора являются: пакеты, каналы, очереди, узлы, изображения узлов, агенты.

Ниже представлены соответствующие им форматы записей.

Пакеты

Формат записей, относящихся к пакетам.

<Type> -t <time> -e <extent> -s <src> -d <dst> -c <conv> -i <id> -a <attr>

Где Type принимает значения:

h - (Hop) Пакет начал передаваться по линии связи от src_addr к dst_addr.

r - (Receive) Пакет, достиг приемника и начал обрабатываться получателем.

d - (Drop) Пакет, отброшен из очереди или линии связи от src_addr к dst_addr.

+ - (Enter queue) Пакет поставлен в очередь линии от src_addr к dst_addr.

- - (Leave queue) Пакет покинул очередь линии от src_addr к dst_addr.

Потери пакетов в линии связи и очереди не различаются. Решение принимается по времени отбрасывания.

Флаги имеют следующие значения:

-t <time> время, наступления события.

-e <extent> размер пакета (в байтах).

-s <src> исходный узел.

-d <dst> узел назначения.

-c <conv> идентификатор соединения.

-i <id> идентификатор пакетов соединения.

-a <attr> атрибут пакета (используется для указания цвета).

Дополнительные флаги, используемые некоторыми протоколами:

-P <pkt_type> задает ASCII строку, определяющую разделенный запятыми список типов пакетов. Например: TCP - пакет tcp данных, ACK - пакет подтверждения (acknowledgement), NACK - пакет отрицательного подтверждения (negative acknowledgement), SRM - пакет SRM данных.

-n <sequence number> последовательный номер пакета (sequence number).

Линии связи (Link)

Записи, относящиеся к линиям связи имеют формат:

l -t <time> -s <src> -d <dst> -S <state> [-c <color>] [-r <bw> -D <delay>]

Поле -S <state> - характеризует изменение состояния линии связи и принимает значения:: UP – восстановление, DOWN – разрыв, COLOR -изменение цвета линии связи. При задании “COLOR”, дополнительный ключ -c <color> устанавливает новый цвет линии. Поля [-r <bw> -D <delay>], указывающие, соответственно, пропускную способность и задержку распространения в канале, используются только при создании соединения.

Очереди (Queue)

Записи, относящиеся к очередям имеют формат:

q -t <time> -s <src> -d <dst> -a <attr>

Ключ -a <attr> - определяет ориентацию отображения очереди, т.е. угол между изображением очереди и линией связи.

Состояние узлов (node)

Записи, относящиеся к изменению состояния узлов, имеют следующий формат:

n -t <time> -s <src> -S <state> [-c <color>] [-o <color>] [-A <labels>]

Флаги -t, -S и -c имеют те же значения, что и для линий связи. Флаг '-A' используется для добавления заданной строки к метке узла. Может использоваться для указания состояния узла. Флаг '-o' используется для восстановления предыдущего цвета отображения узла.

Изображение узла (Mark)

Записи, относящиеся к изображению узлов имеют вид:

Создание изображения узла.

m -t <time> -n <mark name> -s <node> -c <color> -h <shape> [-o <color>]

При этом узел отображается цветным кругом, квадратом или шестиугольником (флаг –h <shape>)

.

Удаление изображения узла

m -t <time> -n <mark name> -s <node> -X

Изменить форму уже созданного узла нельзя.

Агенты

Состояние протокола в аниматоре отслеживается с помощью соответствующих агентов. Записи, относящиеся к агентам имеют формат:

Создание агента:

a -t <time> -n <agent name> -s <src> -d <dst>;

Удаление агента:

a -t <time> -n <agent name> -s <src> -d <dst> -X;

Для отображения состояния переменной протокола используется конструкция 'feature', имеющая следующий формат:

f -t <time> -a <agent name> -T <type> -n <var name> -v <value> -o <prevvalue>

Это позволяет отобразить новое <value> и/или предыдущее <prevvalue> значения переменной <var name> типа <type> агента <agent name>.

В исходном файле аниматора, помимо перечисленных выше, могут присутствовать записи и других форматов. К ним, например, относятся:

Отображение комметария

Запись имеет формат:

v -t <time> TcL script string

Комментарий может включать обычную TcL строку не более 256 символов.

Определение цвета

Запись имеет формат:

c -t <time> -i <color id> -n <color name>

После определения, обращение к цвету может осуществляться по его идентификатору <color id>

Фрагмент Trace-файла .nam приведен на рис. 2.2.

```
c -t * -i 2 -n Red
c -t * -i 3 -n Green
n -t * -a 4 -s 4 -S UP -v circle -c black
n -t * -a 0 -s 0 -S UP -v circle -c black
n -t * -a 5 -s 5 -S UP -v circle -c black
n -t * -a 1 -s 1 -S UP -v circle -c black
n -t * -a 2 -s 2 -S UP -v circle -c black
n -t * -a 3 -s 3 -S UP -v circle -c black
```

Рис. 2.2

3. Утилита построения графиков Xgraph

3.1. Назначение

Утилита Xgraph предназначена для графического отображения функций в декартовой системе координат, при работе в операционной среде X-Windows/UNIX. Программа Xgraph позволяет считывать исходные данные из файлов или непосредственно со стандартного потока ввода. Одновременно может отображаться до 64 независимых функций, используя различные цвета и/или различные стили линий для каждой функции. При отображении графиков доступен вывод заголовков, меток осей, линий разметки или специальных отметок и условных обозначений. Подробное описание утилиты Xgraph приведено в [5].

Отметим, что утилита xgraph разрабатывалась независимо от программы симулятора NS2 и представляет собой универсальное средство графического отображения функций. Поэтому использование xgraph для отображения результатов моделирования симулятора NS2 требует дополнительной обработки trace-файла симулятора для подготовки исходного файла xgraph требуемого формата.

3.2 Формат исходных данных

Входные данные для xgraph задаются в виде текстового файла или стандартного потока ввода. Исходный файл (поток) может содержать один или несколько наборов парных значений данных. Наборы данных разделяются пустой строкой в одном файле. При подключении нескольких входных файлов предполагается, что в каждом файле записан отдельный набор данных (один или несколько). Набор данных состоит из упорядоченного списка точек в форме

"{directive} X Y",

где поле 'directive' принимает значения "draw" или "move" или опущено.

В первом случае строится линия между предыдущей и текущей точками (в случае использования линий при построении графика). При выборе команды "move" линии между точками не строятся. Если команды "draw" или "move" опущены, то по умолчанию

используется "draw" для всех наборов данных, пока не встретиться набор с указанной командой "move".

Имя набора данных может быть указано совместно с данными в строке, заключенной в двойные кавычки (вторые кавычки не обязательны). Опции вывода могут быть определены в самом файле в виде "<option>: <value>". Опции и значения должны быть обязательно разделены пробелом. Ниже представлен пример входного файла с тремя наборами данных. Набор 3 не имеет имени, во втором наборе представлен набор независимых данных, а заголовок графика указан в начале файла.

```
TitleText: Sample Data
0.5 7.8
1.0 6.2
"set one
1.5 8.9

"set two"
-3.4 1.4e-3
-2.0 1.9e-2
move -1.0 2.0e-2
-0.65 2.2e-4

2.2 12.8
2.4 -3.3
2.6 -32.2
2.8 -10.3
```

3.3 Пользовательский интерфейс

Запуск утилиты xgraph осуществляется командой

xgraph [*options*] <*file1*> ... <*fileN*>,

где

options – опции (настройки), управляющие выводом результатов.

<*file1*> ... <*fileN*> - один или несколько файлов исходных данных.

Перечень основных опций программы xgraph приведен в разделе 3.4.

После прочтения исходных файлов xgraph создает окно для графического отображения входных данных. Внешний вид окна Xgraph представлен на Рис. 3.1

Рис. 3.1

После создания окна, все наборы данных отображаются графически с условными обозначениями, показанными в верхнем правом углу окна. Для увеличения части отображаемого графика следует выделить ее в окне xgraph, после чего она автоматически будет отображена в новом окне.

В верхнем левом углу окна помещены кнопки управления: Close, Hardcopy и About.

Close закрывает текущее окно xgraph.

Hardcopy используется для сохранения копии окна в файл или вывод его на печать. При нажатии на нее появляется всплывающее меню, в котором можно установить ряд параметров вывода.

"Output Device" - определяет формат вывода изображения ("HPGL", "Postscript", и т.д.)..

"Disposition" - определяет выходное устройство. При выборе "To Device", необходимо указать имя устройства вывода. При указании "To File" необходимо указать имя выходного файла.

"Maximum Dimension" - определяет максимальный размер графика в сантиметрах. При этом изображение будет масштабировано так, чтобы его максимальный размер не превосходил заданной величины. Если указанное устройство это поддерживает, то изображение может быть повернуто на странице для соответствия заданному размеру.

"Include in Document" - если эта опция установлена, то xgraph будет выводить результаты с возможностью включения их в другие большие документы. Например, при установке этой опции, вывод в Postscript формате будет осуществляться в форме, пригодной для использования с psfig.

"Title Font Family" - определяет шрифт, используемый для отображения заголовка графика. Значения, установленные по умолчанию, пригодны для большинства устройств. Для специфического оборудования необходимо обратиться к его описанию.

"Title Font Size" - определяет требуемый размер шрифта заголовков в точках (1/72 дюйма). Если устройство поддерживает масштабируемые шрифты, то шрифт будет масштабирован до требуемого размера.

"Axis Font Family and Axis Font Size" – определяет тип и размер шрифта меток осей и условных обозначений. (Аналогично "Title Font Family" и "Title Font Size")

После определения всех параметров, нажатие кнопки "Ok" приведет к сохранению копии окна (или выводу на принтер).

About выдает справочную информацию о программе.

Anim- задает режим анимации графика

Replot – осуществляет повторную прорисовку графика

Deriv – строит график производной от исходной функции

3.4. Управление выводом

Утилита xgraph допускает использование большого числа опций. Большинство опций может быть указано, как в командной строке вызова, так и в конфигурационных файлах пользователя .Xdefaults или .Xresources, либо в файле исходных данных..

При задании опций в файле используется имя опции, указанное в скобках.

Перечень основных опций программы xgraph .

\-geometry WxH+X+Y или \=WxH+X+Y (Geometry) - определяет начальное положение и размер окна xgraph;

\-<*digit*> <*name*> - задает имя <*name*> набору данных с номером. <*digit*> (Значение *digit* должно быть в пределах от 0 до 63). Это имя будет использовано в условных обозначениях.

\-bar (BarGraph) - при указании этой опции, будут отображаться линии, начиная от точек данных до базовой точки, определяемой в опции -brb. Обычно вместе с этой опцией используется флаг -nl.

\-device <name> - задает формат выходных данных. По умолчанию, установлено 'X' , другими доступными форматами являются 'ps', 'hpgl', 'idraw' и ‘tgif.’

\-o <filename> (\-O <filename>) - устанавливает имя выходного файла для форматов ps, hpgl и idraw. Файлы, создаваемые с опцией –o, могут быть напечатаны напрямую. Файлы, создаваемые с опцией –O, кроме того, могут использоваться в других документах.

\-P <printername> - устанавливает имя принтера для форматов postscript или hpgl.

\-stk – отображает совпадающие точки различных наборов данных в виде стэка.

\-fitx – задает масштабирование x-координат всех наборов данных к интервалу [0..1].

\-fity – задает масштабирование y-координат всех наборов данных к интервалу [0..1].

\-scale <factor> - задает выходной масштабный множитель для устройств postscript,

hpgl и idraw. По умолчанию - 1.0. При установке в 0.5 будет сгенерировано изображение с размером 50 % от исходного.

\-fmtx <printf-format> \-fmty <printf-format> - устанавливает определенный формат отображения x или y осей.

\-bb (BoundBox) - построение прямоугольников вокруг отображаемых данных. Полезно при использования меток вместо линий для отображения данных (см. опцию -tk).

\-bd <color> (Border) - определяет цвет окантовки для окна xgraph.

\-bg <color> (Background) -определяет цвет фона для окна xgraph.

\-brb <base> (BarBase) - определяет основу для графика с использованием прямоугольников. По умолчанию, <base>=0 , т.е. прямоугольники отображаются от оси X до точек данных.

\-brw <width> (BarWidth) - определяет ширину прямоугольников. По умолчанию отображаются прямоугольники шириной один пиксель.

\-bw <size> (BorderSize) – задает ширину окантовки окна xgraph.

\-db (Debug) – выводит значения всех установленных опций..

\-fg <color> (Foreground) – задает цвет переднего плана, которым отображаются все линии и текст в xgraph.

\-gw <GridSize> - задает ширину линий разметки. в пикселях

\-gs <GridStyle> - задание стиля отображаемых линий разметки.

\-lf <fontname> (LabelFont) - задает шрифт меток. Все метки осей и линий разметки будут отображаться с использованием этого шрифта. Имя шрифта может быть отображено прямо в соответствии с его типом (т.е. "9x15" или "-*-courier-bold-r-normal-*-140-*"), либо в форме аббревиатуры <family>-<size>, family - семейство шрифтов (например - helvetica) и

size - размер шрифта в точках (например - 12). Значение по умолчанию для этого параметра - "helvetica-12".

\-lnx (LogX) – задает отображение оси X в логарифмическом масштабе. Разметка оси отображает показатель степени числа 10.

\-lny (LogY) – задает отображение оси Y в логарифмическом масштабе. Разметка оси отображает показатель степени числа 10.

\-lw <width> (LineWidth) - определяет ширину линий отображения данных в пикселях. По умолчанию - ноль.

\-lx <xl,xh> (XLowLimit, XHighLimit) - ограничивает диапазон оси X заданным интервалом. Вместе с опцией -ly используется для изображения крупным планом интересуемых фрагментов больших графиков.

\-ly <yl,yh> (YLowLimit, YHighLimit) - ограничивает диапазон оси Y заданным интервалом.

\-m (Markers) - маркирует каждую точку данных различным маркером. В xgraph имеется восемь типов маркеров. Маркеры одного типа могут отличаться цветом.

\-M (StyleMarkers) – в отличие от опции -m, присваивает специфичный маркер каждому набору данных.

\-nl (NoLines) - отключает отображение линий. Вместе с опциями -m, -M, -p, или -P используется для отображения точечных графиков. Вместе с опцией -bar используется для построения стандартных графиков на основе прямоугольников.

\-ng (NoLegend) - отключает отображения условных обозначений. Служит для увеличения полезной площади отображения.

\-p (PixelMarkers) - маркирует каждую точку данных небольшим маркером (пиксельного размера). Обычно используется вместе с опцией -nl для построения точечных графиков.

\-P (LargePixels) - аналогично –p, но используются крупные маркеры.

\-rv (ReverseVideo) – инвертирует цвет отображения на черно-белых мониторах.. Для цветных мониторов опция не определена.

\-t <string> (TitleText) – задает заголовок графика. Строка - заголовок будет отображена в центре сверху графика.

\-tf <fontname> (TitleFont) – задание шрифта заголовка. Формат аналогичен опции \-lf. По умолчанию задан шрифт "helvetica-18".

\-tk (Ticks) - отображение данных с помощью маркеров, а не линий. Удобно использовать вместе с опцией -bb.

\-tkax (Tick Axis) - отображает оси при использовании маркеров.

\-x <unitname> (XUnitText) - определяет имя (надпись) оси X. По умолчанию "X".

\-y <unitname> (YUnitText) - - определяет имя (надпись) оси Y. По умолчанию "Y".

\-zg <color> (ZeroColor) - определяет цвет отображения нулевой линии сетки координат

\-zw <width> (ZeroWidth) – задает ширину нулевой линии сетки координат (в пикселях).

Часть опций может быть указана только в конфигурационном файле среды X `defaults` или в файлах данных. Эти опции описаны ниже.

<digit>.Color - определяет цвет набора данных. Всего определено восемь цветов, т.е. digit может принимать значение между '0' и '7'. При использовании более восьми наборов данных, цвета наборов будут повторяться, но будет изменяться тип линий отображаемых данных.

<digit>.Style - определяет тип линий для набора данных. В xgraph также

присутствует восемь типов линий для отображения данных (digit может принимать значение между '0' и '7'). При использовании более восьми наборов данных тип линий наборов будут повторяться. Таким образом, при использовании цветных мониторов, возможно отображение 64 различных наборов данных.

Device - задает формат вывода изображения по умолчанию в диалоговом окне hardcopy (т.е. "Postscript", "HPGL", и т.д.).

Disposition - задает значение устройства вывода по умолчанию в диалоговом окне hardcopy. Возможные значения "To File" или "To Device".

FileOrDev - задает значение имени файла или устройства вывода по умолчанию в диалоговом окне hardcopy.

ZeroWidth – задает ширину нулевой осевой линии в пикселях.

ZeroStyle - определяет цвет нулевой осевой линии.

4 Практическое освоение симулятора

В данном разделе предложен поэтапный метод освоения возможностей симулятора, на основе серии лабораторных работ возрастающей сложности.
Методические указания предполагают предварительное знакомство учащихся с описанием симулятора NS2 , утилит NAM и XGRAPH (разделы 1-3), а также базовыми командами ОС UNIX , пользовательским интерфейсом X-Windows, и каким-либо текстовым редактором ОС UNIX.

4.1 Освоение возможностей симулятора

Данный цикл работ охватывает изучение возможностей симулятора по моделированию различных аспектов сетевого взаимодействия.

Работа №1 Ознакомление с принципами работы симулятора NS2

1. Цель работы

- практическое ознакомление с работой симулятора;
- приобретение навыков по созданию моделей сетей различной топологии, а также сетей с различными характеристиками каналов;
- освоение пользовательского интерфейса утилиты визуального отображения результатов моделирования nam (**N**etwork **A**ni**m**ator).

2. Порядок работы с симулятором

При подготовке к работе ознакомиться с описанием симулятора NS2, его системой команд и основными режимами работы.

В симуляторе NS2 исходные данные для моделирования задаются с помощью программной модели - скрипта, использующей командный язык симулятора, поэтому, в общем виде, процесс моделирования в симуляторе NS2 включает:

Создание скрипта в соответствии с целью моделирования. Сохранение его в виде текстового файла в формате .tcl (рекомендуется).

Событийное моделирование (запуск симулятора) в соответствии с созданным скриптом.

Анализ и обработку результатов моделирования.

2.1. Создание скрипта

Перед созданием скрипта необходимо перейти в каталог, в котором будут находиться ваши рабочие файлы, это можно сделать командой

'cd <dir>',

где '<dir>' - путь к вашему рабочему каталогу.

Скрипт может быть создан в любом текстовом редакторе (например, редакторе xedit) и сохранен в виде файла с расширением tcl.

Редактор можно запустить командой

'xedit <work1_1.tcl>',

где '<work1_1.tcl>' - имя создаваемого скрипта.

2.1.1. Общие требования к скрипту

Скрипт пишется на командном языке симулятора, являющемся своеобразным расширением языка TCL (Tool Command Language). Список основных команд и особенности языка моделирования представлены в главе 1.

Простейший скрипт должен включать описания:

- топологии сети, включающей перечень узлов и характеристики каналов связи,
- характеристик потоков данных и используемых протоколов
- хронологии внешних событий (открытие и закрытие сеансов связи, отказы, моменты начала и конца моделирования и т.п.)
- формата вывода результатов моделирования.

В зависимости от целей моделирования, скрипт может содержать описания протоколов, различных уровней взаимодействия, используемых в узлах сети, характеристики потоков отказов, модели помех и т.д.

2.1.2. Пример создания простейшего скрипта

Пусть моделируемая сеть (Рис.4.1) состоит из двух узлов: s1, r1, и дуплексного канала связи с пропускной способностью 2 Мбит/с, задержкой распространения 5 ms и механизмом обслуживания очереди FIFO.

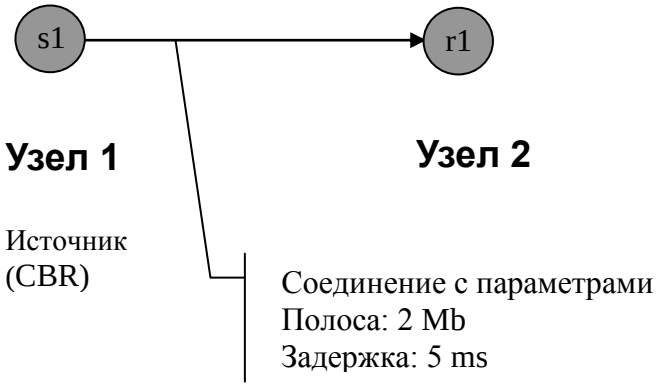


Рис. 4.1

В узле s1 находится источник данных, генерируемых с постоянной скоростью (CBR), с параметрами: размер пакетов 200 байт, период следования пакетов 0.005 сек. В узле r1 находится приемник трафика. Источник трафика запускается через 0.5 сек после начала моделирования и прекращает работу через 4.5 сек. Моделирование продолжается 5.0 сек. Результаты моделирования должны быть выведены в выходной файл данных и отображены с помощью утилиты nam.

Текст скрипта

Создаем новый объект класса Simulator

```
set ns [new Simulator]
```

Далее, откроем файл, который будет использоваться для записи выходных результатов моделирования с последующей обработкой утилитой (nam).

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

Первая строка открывает файл out.nam для записи и присваивает ему управляющую переменную nf. Вторая строка указывает симулятору записывать все данные о динамике модели в этот файл.

Определим процедуру 'finish', которая закрывает trace файл и запускает утилиту nam.

```
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam
        exit 0
}
```

Создаем объекты типа узел и присваиваем им имена s1 и r1.

```
set s1 [$ns node]
set r1 [$ns node]
```

Создаем двунаправленную линию связи между узлами s1 и r1 с пропускной способностью 2 Мб/с, задержкой распространения 5 ms и механизмом обслуживания очереди FIFO.

```
$ns duplex-link $s1 $r1 2Mb 5ms DropTail
```

Определяем узел s1, как источник CBR трафика. Для этого создаем CBR-агента с заданными характеристиками и прикрепляем его к узлу s1.

```
set cbr1 [new Agent/CBR]
$ns attach-agent $s1 $cbr1
$cbr1 set packetSize_  200
$cbr1 set interval_  0.005
```

Создаем простейший агент-приемник (заглушку) и прикрепляем его к узлу r1.

```
set null1 [new Agent/Null]
$ns attach-agent $r1 $null1
```

Соединяем источник и приемник трафика.

```
$ns connect $cbr1 $null1
```

Задаем режим работы (график событий) симулятора.

Начало и конец работы CBR-агента:

```
$ns at 0.5 "$cbr1 start"
$ns at 4.5 "$cbr1 stop"
```

Вызов процедуры "finish" после заданного интервала моделирования.

```
$ns at 5.0 "finish"
```

Команда запуска симулятора.

```
$ns run
```

Записав приведенный скрипт в файл 'work1.tcl', можно переходить к запуску симулятора.

2.2. Запуск симулятора

Запуск модели в симуляторе ns осуществляется командой

'ns <tclscript>',

где '<tclscript>' - имя файла (с расширением .tcl), определяющего сценарий моделирования. Предполагается, что Вы находитесь в директории с исполняемыми файлами ns, или ваш путь указывает на эту директорию.
Запустить симулятор для исполнения приведенного выше скрипта можно командой

ns work1.tcl

После выполнения сценария, результаты работы симулятора будут записаны в trace-файл out.nam, который является входным файлом для утилиты визуального отображения nam (Network AniMator). Окно аниматора nam появится после выполнения скрипта, в нем будет отображена моделируемая топология и происходящие события – передача пакетов.

2.3. Анализ и обработка результатов моделирования

Данный пример позволяет визуально проанализировать результаты моделирования в окне утилиты nam. Клавиши меню окна аниматора позволяют прокручивать реализацию модели с любого момента времени, изменять скорость моделирования, а также ориентацию и размер графа сети. Щелкнув указателем мыши по компоненту сети или пакету, можно получить дополнительную информацию об указанном объекте.

Проанализировать результаты моделирования можно также путем просмотра регистрационного файла симулятора в окне текстового редактора.

3. Задание к работе

3.1. Исходное задание

Создайте базовую модель сети в соответствии с вариантом задания (Рис.4.2.). Данная модель сети будет использоваться и в других работах.

Сеть включает одиннадцать узлов: s1-s3, r1 - r5, k1 - k3.

Все каналы связи имеют задержку распространения 5 миллисекунд. Канал r1-r2 имеет пропускную способность 1Мб/с, все остальные - 2 Мб/с .

Опция автоматического расположения объектов при отображении топологии сети в nam не всегда позволяет нужным образом ориентировать соединения сети. Для корректного отображения объектов аниматора необходимо явно задать их расположение (в этом случае, опция re-layout будет недоступна).

Расположить звено передачи требуемым образом можно с помощью команды:

$ns duplex-link-op $node1 $node2 orient right,

которая ориентирует линию связи между узлами node1 и node2 горизонтально направо (опция right), или, например,

$ns duplex-link-op $node3 $node4 orient right-up

- звено связи node3-node4 будет расположено под углом 45 градусов (направо и вверх). Всего используются четыре опции - right, left, up, down, а также их комбинации.

Отметим, что при отображении топологии сети в аниматоре, длина каждой отображаемой линии связи определяется симулятором с учетом ее пропускной способности и задержки распространения. Поэтому, в ряде случаев, картинка топологии сети может отличаться от ожидаемой.

После задания топологии сети, установите в сети агенты - источники и приемники трафика. Узел s1 определите источником трафика, генерируемого с постоянной скоростью (CBR), а узел k1 - приемником трафика.

Параметры CBR-источника: размер пакетов 300 байт, период следования 0.005 сек.

Источник трафика запускается через 0.1 сек после начала моделирования и прекращает работу через 5.0 сек. Моделирование продолжается 8.0 сек.

В скрипте необходимо описать топологию данной сети, определить команды запуска и остановки источника трафика в заданное время, вывести результаты работы в выходной файл данных и отобразить их с помощью утилиты nam.

3.2. Модификация исходной топологии

В построенном варианте сетевой модели пропускная способность каналов вполне достаточна для передачи трафика между источником и приемником. Измените параметры CBR-источника следующим образом:

размер пакетов источника трафика - 300 байт,

период следования пакетов - 0.002 сек.

Текст скрипта сохраните в файле work1_2.tcl.

Простые вычисления показывают, что указанная пропускная способность недостаточна для передачи CBR-трафика с заданными параметрами. Часть пакетов будет теряться.

После запуска симулятора с модифицированным сценарием, потерю пакетов можно проследить с помощью сгенерированного trace файла out.nam. Открыв его в текстовом редакторе, можно убедиться, что часть пакетов действительно отброшена (они маркируются символом 'd' - drop; формат trace файла приведен в описании симулятора).

3.3. Мониторинг очереди соединения

Чтобы отобразить процесс заполнения очередей в аниматоре nam, необходимо задать команду мониторинга очереди соответствующего соединения. Фрагмент модифицированного текста скрипта выглядит следующим образом:

…

Создаем двунаправленное соединение между узлами.

```
$ns duplex-link $r1 $r2 1Mb 5ms DropTail
$ns duplex-link-op $r1 $r2 orient right
```

Устанавливаем вертикальное отображение очереди соединения между узлами r1 и r2.

```
$ns duplex-link-op $r1 $r2 queuePos 0.5
```

Вводим ограничение максимального размера очереди - . 20 пакетов

```
$ns queue-limit $r1 $r2 20
```

…

Модифицируйте скрипт подобным образом, добавив мониторинг соединения между узлами r1 и r2 и установив размер очереди в 10 пакетов (файл work1_3.tcl). После запуска скрипта, в окне аниматора можно наблюдать заполнение очереди указанного соединения.

Через некоторое время произойдет переполнение очереди, и отбрасываемые пакеты будут также отображаться в аниматоре.

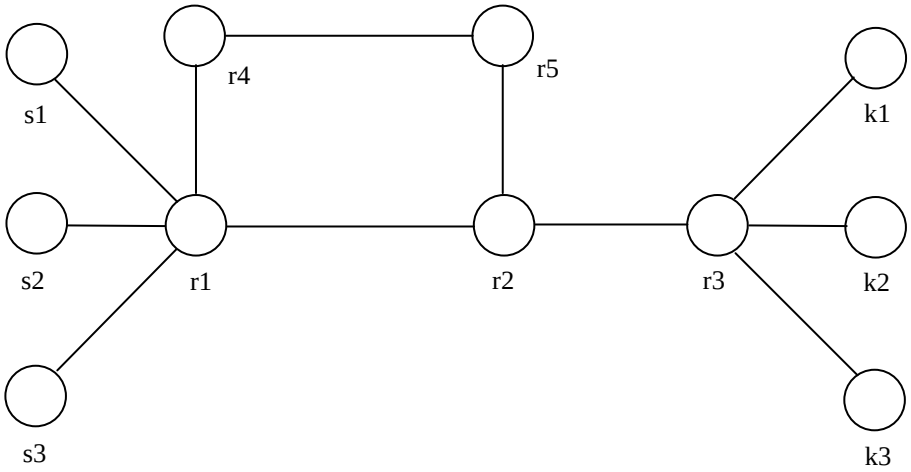
3.4. Устранение потери пакетов

Модифицируйте скрипт work1_3.tcl, изменив пропускную способность каналов таким образом, чтобы ее было достаточно для передачи CBR-трафика без потерь (файл work1_4.tcl).

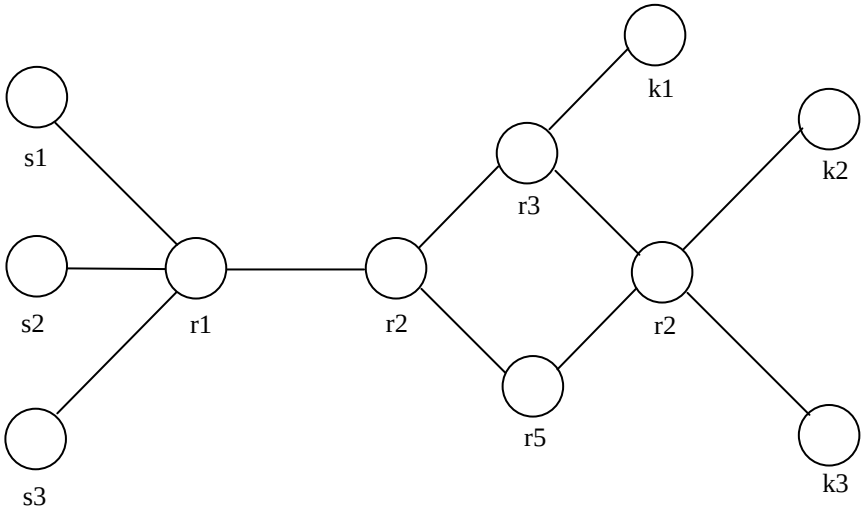
4. Программа работы

1. Создайте файл work1_0.tcl с приведенным выше текстом простейшего скрипта (пункт 2.1.2), запустите скрипт на моделирование.
2. Ознакомьтесь с действием управляющих кнопок аниматора nam, в том числе, с опцией автоматического расположения объектов (re-layout).
3. Создайте скрипт в соответствии с заданием (п.3.1), сохраните его в виде файла work1_1.tcl и запустите симулятор. Проанализируйте результаты моделирования.
4. Измените скрипт в соответствии с п.п. 3.2 - 3.3, сохраните модель в виде файлов work1_2.tcl, work1_3.tcl и запустите симулятор, Убедитесь в наличии отбрасываемых пакетов путем просмотра выходного файла out.nam в текстовом редакторе, а также с помощью команды мониторинга очереди и утилиты nam.
5. Модифицируйте скрипт в соответствии с п. 3.4., убедитесь в отсутствии отброшенных пакетов.

Вариант 1



Вариант 2



Вариант 3

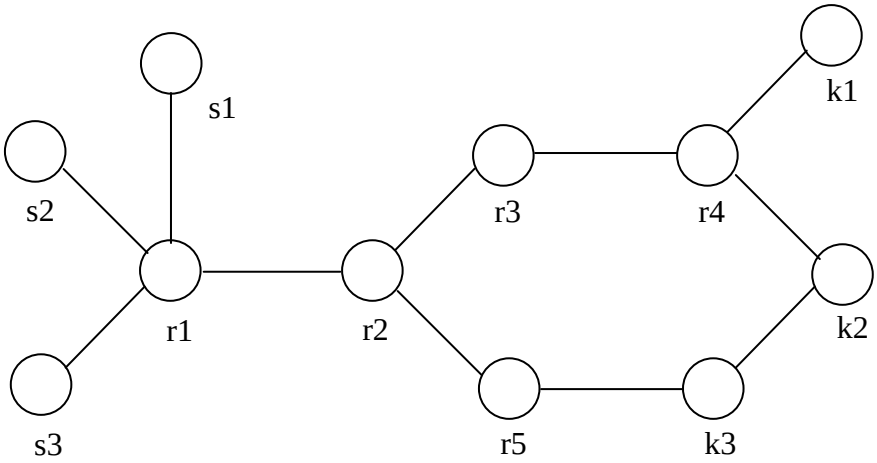


Рис. 4.2 Варианты заданий

Работа №2 Моделирование сетей с несколькими потоками и различными характеристиками источников трафика

1. Цель работы

- приобретение навыков моделирования сетей с несколькими потоками данных от различных источников;
- ознакомление с возможностями симулятора по моделированию потоков данных типовых интернет-приложений;
- ознакомление со способами моделирования потоков данных с заданными статистическими характеристиками источников трафика.

2. Особенности моделирования сетей с несколькими потоками данных

При моделировании сетей с несколькими потоками данных для каждого потока следует указать:
- источник трафика;
- приемник трафика;
- статистические (временные) характеристики источников данных;
- конкретную реализацию протокола, используемого на транспортном уровне.

Кроме того, при использовании утилиты визуального отображения nam, для различения потоков целесообразно “пометить” пакеты различных потоков разными цветами.

При моделировании типовых интернет-приложений следует использовать соответствующий данному приложению агент, учитывающий вероятностные характеристики конкретного приложения. Перечень агентов типовых интернет-приложений с указанием конфигурационных параметров приведен в описании симулятора. При моделировании специфичных приложений статистические характеристики источника должны быть заданы явно.

2.1 Пример модели сети с двумя потоками

Пусть моделируемая сеть имеет вид, показанный на рис. 4.3.

Сеть объединяет четыре узла: s1, s2, r1 и r2. Все линии связи имеют пропускную способность 128 кбит/сек и задержку распространения 100 мсек.

В сети одновременно работают два типовых интернет-приложения. Между узлами s1 и r2 работает протокол ftp (причем, узел s1 является источником ftp-трафика, а узел r2 - приемником), а между узлами s2 и r2 - прикладной протокол telnet, (причем, s2 является источником telnet-трафика, а r2 - приемником). Узел r1 является промежуточным (маршрутизатором) и не является ни источником, ни приемником трафика.

Оба приложения используют на транспортном уровне протокол TCP. Параметры протокола TCP для обоих приложений одинаковы:

- размер пакетов 100 байт;
- максимальный размер окна насыщения - 50 пакетов.

Сеанс FTP характеризуется размером передаваемого файла - 175 пакетов.

Трафик telnet характеризуется средним интервалом между генерируемыми пакетами - 0.03 сек. Источник ftp-трафика запускается через 0.1 сек после начала моделирования и прекращает работу после генерации 175 пакетов.

Сеанс telnet запускается через 0.5 сек и заканчивается через 1.5 сек после начала моделирования.

Моделирование продолжается 6.0 сек.

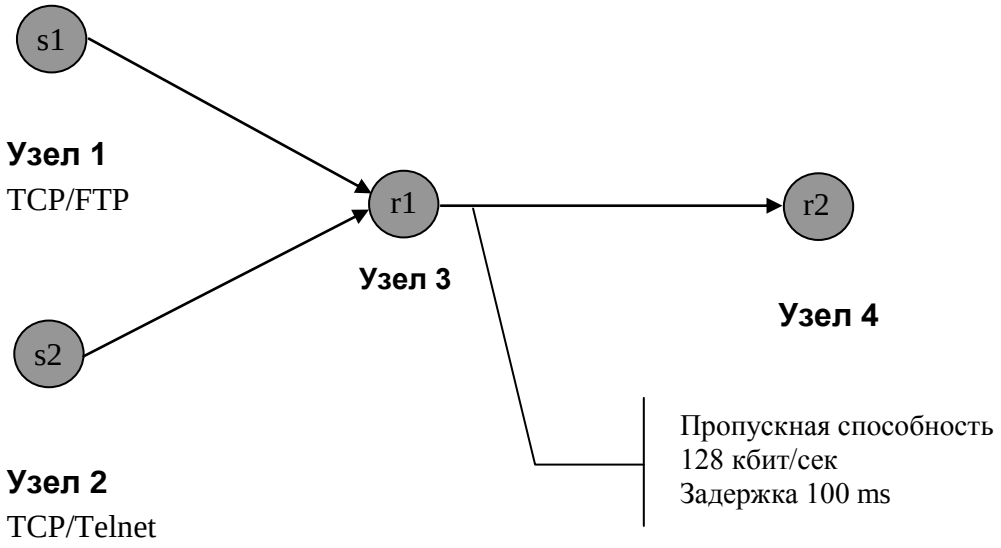


Рис. 4.3

Рассмотрим скрипт, моделирующий описанную сеть и отображающий результат с помощью утилиты nam. При его создании удобно использовать в качестве шаблона скрипт, созданный в работе №1.

Текст скрипта

```
set ns [new Simulator]

$ns color 1 Blue
$ns color 2 Red

set nf [open out.nam w]
$ns namtrace-all $nf

proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam
        exit 0}

set s1 [$ns node]
set s2 [$ns node]
set r1 [$ns node]
set r2 [$ns node]

$ns duplex-link $s1 $r1 128kb 100ms DropTail
$ns duplex-link $s2 $r1 128kb 100ms DropTail
```

```
$ns duplex-link $r1 $r2 128kb 100ms DropTail

$ns duplex-link-op $s1 $r1 orient right-down
$ns duplex-link-op $s2 $r1 orient right-up
$ns duplex-link-op $r1 $r2 orient right

$ns queue-limit $r1 $r2 10

$ns duplex-link-op $r1 $r2 queuePos 0.5

set snk1 [new Agent/TCPSink]
$ns attach-agent $r2 $snk1

set snk2 [new Agent/TCPSink]
$ns attach-agent $r2 $snk2

set tcp1 [new Agent/TCP]
$tcp1 set maxcwnd_ 50
$tcp1 set packetSize_ 100
$ns attach-agent $s1 $tcp1
$ns connect $tcp1 $snk1
$tcp1 set fid_ 1
set ftp1 [$tcp1 attach-source FTP]

set tcp2 [new Agent/TCP]
$tcp2 set maxcwnd_ 50
$tcp2 set packetSize_ 100
$ns attach-agent $s2 $tcp2
$ns connect $tcp2 $snk2
$tcp2 set fid_ 2
set tln1 [$tcp2 attach-source Telnet]
$tln1 set interval_ 0.03s

$ns at 0.1 "$ftp1 produce 175"
$ns at 0.5 "$tln1 start"

$ns at 1.5 "$tln1 stop"

$ns at 6.0 "finish"

$ns run
```

Для различения пакетов источников tcp1 и tcp2 им были установлены, соответственно, синий и красный цвета, что было сделано командами $tcp1 set fid_ 1 и $tcp2 set fid_ 2, где 1 и 2 - идентификаторы цветов. При отображении в nam пакеты ftp источника будут отображаться синим цветом, а пакеты генерируемые telnet источником - красным. В скрипте установлен мониторинг очереди соединения между узлами r1 и r2. По умолчанию TCP-агент использует версию протокола TCP-Tahoe. Использование агентов прикладного и транспортного уровней понятно из текста скрипта.

2.2 Задание к самостоятельной работе

Создайте следующую модель сети, используя, базовую топологию вашего варианта задания работы 1.

Все линии связи сети обладают полосой пропускания 128 кбит/сек и задержкой распространения 10 мсек.

Между узлами s1 и k1 работает протокол ftp (причем, узел s1 является источником ftp-трафика, а узел k1 - приемником), а между узлами s2 и k2 - прикладной протокол telnet, (причем, s2 является источником telnet-трафика, а k2 - приемником). Узлы r1 - r5 являются промежуточными и не является ни источниками, ни приемниками трафика.

Оба приложения используют на транспортном уровне протокол TCP-Tahoe. Параметры протокола TCP для обоих приложений одинаковы:

- размер пакетов - 100 байт;
- максимальный размер окна насыщения - 50 пакетов.

Сеанс FTP начинается через 0.1 сек после начала моделирования и характеризуется размером передаваемого файла - 100 пакетов.

Сеанс telnet запускается через 0.2 сек со средним интервалом между генерируемыми пакетами - 0.02 сек, и заканчивается через 2.0 сек после начала моделирования.

В скрипте укажите команду мониторинга очереди в канале r1-r2, по которому проходит трафик обоих источников. Установите размер очереди равным 10 пакетов.

Моделирование продолжается 8.0 сек.

Результаты моделирования отобразите с помощью утилиты nam.

3. Моделирование потоков с заданными статистическими характеристиками

Для моделирования потоков с заданными статистическими характеристиками в симуляторе предусмотрены специальные агенты прикладного уровня, работающие в связке с UDP-агентом транспортного уровня.

Симулятор имеет четыре вида агентов для генерации трафика с заданным распределением:

Traffic/CBR - генерируемый трафик состоит из пакетов заданного размера и имеет постоянную скорость;

Traffic/Expoo - трафик характеризуется экспоненциальным распределением длительности On/Off интервалов (во время 'on' периода трафик генерируется с постоянной битовой скоростью, во время 'off' периодов трафик отсутствует);

Traffic/Pareto - трафик характеризуется Парето распределением On/Off интервалов;

Traffic/Trace - трафик генерируется на основе имеющегося trace-файла данных.

Параметры источников с заданным распределением, а также формат trace-файла приведены в описании к симулятору.

Пример модели потока с экспоненциальным распределением интервалов времени генерации

Создаем UDP агент и прикрепляем его к узлу s1.

```
set udp1 [new Agent/CBR/UDP]
$ns attach-agent $s1 $udp1
```

Создаем агент Expoo трафика и задаем его конфигурационные параметры.

```
set traffic [new Traffic/Expoo]
$traffic set packet-size 100
```

```
$traffic set burst-time 0.002s
$traffic set idle-time 0.002s
$traffic set rate 100k
```

Прикрепляем агент трафика к источнику трафика.

```
$udp1 attach-traffic $traffic
```

Соединяем источник udp1 и приемник snk1. (Заметим, что приемником, в этом случае, должен быть простейший агент-заглушка)

```
$ns connect $udp1 $snk1
```

…

Устанавливаем моменты запуска и останова источника.

```
$ns at 1.0 "$udp1 start"
…
$ns at 5.0 "$udp1 stop"
…
```

С помощью данной процедуры создается UDP агент и прикрепляется к узлу s1. Далее, создается агент экспоненциального трафика и задаются его конфигурационные параметры: размер пакетов - 100 байт, параметры характеризующие экспоненциальное распределение - времена ожидания и генерации idle- и burst-time по 0.002 секунды и максимальная скорость генерации источника 100 килобайт/сек.

Последующими командами агент трафика прикрепляется к агенту UDP, и затем, источник и приемник трафика udp1 и snk1 соединяются между собой. Запуск и остановка генератора осуществляется командами "start" и "stop".

Задание к самостоятельной работе

В имеющейся модели замените трафик приложения telnet на трафик с экспоненциальным распределением. Установите для него следующие параметры:

- burst-time - 0.02 сек;
- idle-time - 0.01 сек, rate - 150k;
- время запуска 0.2 сек,;
- время остановки 3.0 сек;
- размер пакетов 100 байт.

Кроме того, установите следующие параметры FTP приложения:

- число генерируемых пакетов 100,
- время запуска 0.1 сек.

Скрипт сохраните в файле work2_2.tcl.

Использование trace файла данных для генерации трафика

Кроме встроенных генераторов трафика с заданными характеристиками, симулятор позволяет производить моделирование трафика на основе имеющегося файла данных. В качестве такого файла может использоваться trace-файл данных реальной сети. Trace-файл

симулятора содержит информацию о времени между отправкой пакетов и размере отправляемых пакетов (подробнее см. описание симулятора).

Добавить источник подобного типа можно с помощью следующего набора команд:

```
...
set tfile [new Tracefile]
$tfile filename trace1.dat
set src1 [new Agent/CBR/UDP]
$ns attach-agent $n0 $src1

set null1 [new Agent/Null]
$ns attach-agent $n1 $null1

$ns connect $src1 $null1
set trace1 [new Traffic/Trace]
$trace1 attach-tracefile $tfile

$src1 attach-traffic $trace1

...

$ns at 0.0 "$src1 start"

...
```

С помощью приведенного фрагмента скрипта осуществляется создание источника трафика trace1, использующего файл данных trace.dat. После чего источник trace1 устанавливается поверх транспортного UDP-агента src1, который соединяется с приемником трафика типа Null. Запускается источник командой start.

Модифицируйте скрипт задания 3.2, заменив в имеющейся модели сети источник expoo трафика источником трафика, генерируемого на основе trace файла. Скрипт сохраните в файле work2_3.tcl. Используйте файл данных trace1.dat, запуская скрипт из одной с ним директории (либо укажите путь к этому файлу в скрипте).

Несколько источников трафика могут использовать один и тот же trace файл данных. Для того чтобы исключить их синхронизованную работу стартовое место в файле для каждого из источников выбирается произвольно.

Замените в скрипте предыдущего задания источник ftp трафика также источником трафика, генерируемого на основе файла trace1.dat. Скрипт сохраните в файле work2_4.tcl. Время запуска источников 0.1 сек, время остановки 3 сек.

Программа работы

1. Создайте файл work2_0.tcl с приведенным в п. 2.1 текстом скрипта и запустите его на моделирование.
2. Создайте скрипт в соответствии с индивидуальным заданием (п. 2.2), сохраните его в виде файла work2_1.tcl и запустите скрипт на моделирование. Проанализируйте результаты.

3. Модифицируйте скрипт предыдущего задания в соответствии с п. 3.2, сохраните его в виде файла work2_2.tcl и запустите его на моделирование.
4. Измените скрипт в соответствии с п.п 4.1, 4.2 и сохраните результаты в файлах work2_3.tcl и work2_4.tcl соответственно. После запуска скрипта пункта 4.2 убедитесь в том, что источники работают не синхронизованно, хотя используют один и тот же файл данных.

Работа №3 Моделирование сетей с отказами и различными стратегиями маршрутизации

Цель работы

- приобретение навыков моделирования сетей с отказами в каналах связи;
- ознакомление с возможностями симулятора по моделированию процессов маршрутизации;
- исследование методов управления сетевой динамикой.

Моделирование сетей с отказами в каналах связи

В компьютерных сетях нередки случаи отказов в каналах связи. Они могут быть вызваны разрывами физических линий связи, отказами в каналообразующей аппаратуре и другими причинами. Эти отказы, как правило, носят кратковременный характер, и канал через определенное время восстанавливается. Симулятор позволяет моделировать как невосстанавливаемые, так и кратковременные отказы каналов связи и указывать динамику их появления.

Для моделирования наступления отказа в канале используется команда **down** (по отношению к методу rtmodel) с указанием момента времени и конкретного звена передачи данных . Восстановление канала моделируется аналогичной по структуре командой **up.**

По умолчанию, при отображении результатов моделирования утилитой nam, неисправный канал помечается красным цветом.

2.1 Пример модели сети с отказом в канале связи

Пусть моделируемая сеть имеет вид, показанный на Рис. 4.4. Сеть состоит из трех узлов: s1, r1 и r2. Все линии связи обладают полосой пропускания 256 кбит/сек и задержкой распространения 150 мсек.

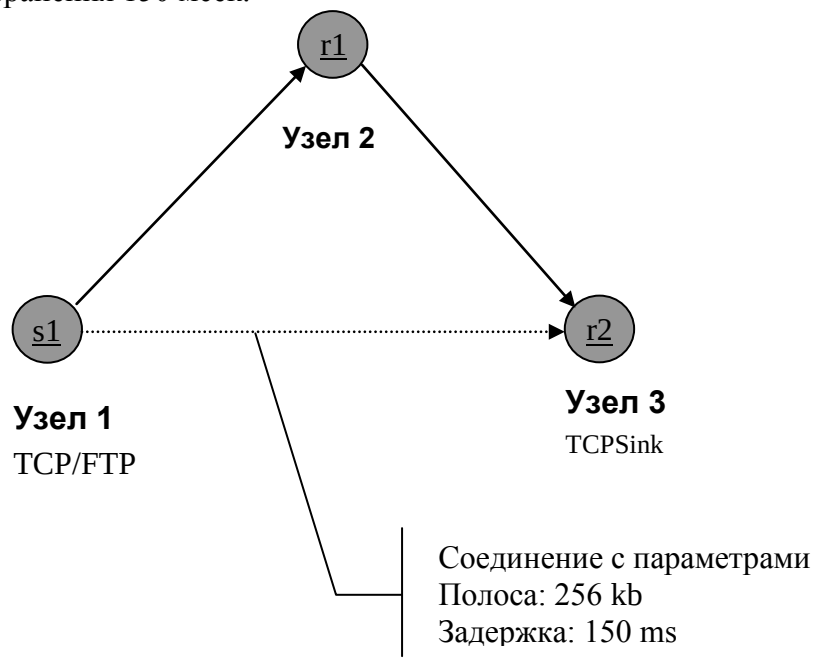


Рис 4.4

Между узлами s1 и r2 работает протокол FTP (в узле s1 находится источник FTP

трафика, прикрепленный к агенту TCP, а в узле r2 - приемник). Узел r1 является промежуточным (маршрутизатором) и не является ни источником, ни приемником трафика.

Приложение FTP использует на транспортном уровне протокол TCP со следующими параметрами:

- размер пакетов 200 байт;
- максимальный размер окна насыщения - 50 пакетов.

Сеанс FTP характеризуется размером передаваемого файла - 70 пакетов.

Источник FTP-трафика запускается через 0.1 сек. после начала моделирования и прекращает работу после генерации 70 пакетов.

Через 2.0 сек после начала моделирования происходит разрыв соединения между узлами s1 и r2, а еще через 1.0 сек - ее восстановление.

Моделирование продолжается 6.0 сек.

Рассмотрим скрипт, моделирующий описанную сеть и отображающий результат с помощью утилиты nam.

Текст скрипта (файл work3_0.tcl)

```
set ns [new Simulator]

$ns color 1 Blue

set nf [open out.nam w]
$ns namtrace-all $nf

proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam
        exit 0}

set s1 [$ns node]
set r1 [$ns node]
set r2 [$ns node]

$ns duplex-link $s1 $r2 256kb 150ms DropTail
$ns duplex-link $s1 $r1 256kb 150ms DropTail
$ns duplex-link $r1 $r2 256kb 150ms DropTail

$ns duplex-link-op $s1 $r2 orient right
$ns duplex-link-op $s1 $r1 orient right-up
$ns duplex-link-op $r1 $r2 orient right-down

$ns queue-limit $s1 $r2 10

set snk1 [new Agent/TCPSink]
$ns attach-agent $r2 $snk1

set tcp1 [new Agent/TCP]
$tcp1 set maxcwnd_ 50
```

```
$tcp1 set packetSize_ 200
$ns attach-agent $s1 $tcp1
$ns connect $tcp1 $snk1
$tcp1 set fid_ 1
set ftp1 [$tcp1 attach-source FTP]

$ns at 0.1 "$ftp1 produce 70"
# Отказ  канала
$ns rtmodel-at 2.0 down $s1 $r2
# Восстановление канала
$ns rtmodel-at 3.0 up $s1 $r2

$ns at 6.0 "finish"

$ns run
```

При отображении утилитой nam, пакеты (TCP-сегменты) источника ftp1 будут помечены синим цветом. Канал связи между узлами s1 и r2 во время разрыва будет отображен красным цветом.

2.2 Задание к самостоятельной работе

Создайте следующую модель сети, используя топологию заданного варианта:

Все линии связи имеют пропускную способность 256 кбит/сек и задержку распространения 20 мсек.

Между узлами s1 и k1 работает протокол ftp (узел s1 является источником ftp-трафика, а узел k1 - приемником). Узлы r1 - r5 являются промежуточными и не является ни источниками, ни приемниками трафика.

Приложение ftp использует на транспортном уровне протокол TCP. Параметры протокола TCP:

- размер пакетов 300 байт;
- максимальный размер окна насыщения - 50 пакетов.

Сеанс FTP начинается через 0.5 сек после начала моделирования и характеризуется размером передаваемого файла - 300 пакетов.

Через 2.0 сек после начала моделирования происходит разрыв линии связи между узлами r1 и r2 (1-ый вариант задания), r2 и r3 (2-ой и 3-ий варианты), а через 3.0 сек ее восстановление. Моделирование продолжается 6.0 сек.

Скрипт сохраните в файле work3_1.tcl.

Моделирование различных стратегий маршрутизации

В симуляторе, по умолчанию, во всех узлах используется статическая модель маршрутизации, при которой маршрут между любыми узлами сети остается неизменным в течение всего сеанса моделирования. Пакеты любого соединения всегда движутся по одному (кратчайшему - по числу хопов) пути. При такой стратегии маршрутизации отказ какого-либо канала связи приводит к фатальной потере пакетов и к разрыву соответствующего соединения.

Симулятор позволяет моделировать различные стратегии маршрутизации, в том числе, такие, при которых в узлах сети возможна динамическая корректировка таблиц маршрутизации, в соответствии с реальной обстановкой.

В симуляторе существуют модели маршрутизации типа Static, Session и DV.

Стратегия Static routing используется по умолчанию. При этом, алгоритм расчета маршрутов в сети запускается единственный раз в начале моделирования и использует “весовые коэффициенты” (цены) всех каналов связи. Если в процессе моделирования происходит изменение топологии сети – отказ канала, то некоторые узлы сети могут оказаться недоступны друг для друга.

Стратегия Session использует тот же алгоритм расчета маршрута, что и Static, но, в отличие от нее, при разрывах/восстановлениях каналов производится соответствующее вычисление новых маршрутов.

Стратегия DV реализует динамическую маршрутизацию, использующую механизм DV-агентов, прикрепленных к узлам сети. Каждый агент периодически, через заданный интервал времени, посылает в сеть информацию о текущем состоянии сети, на основании которой обновляются таблицы маршрутизации в узлах. (Подробнее о протоколах маршрутизации см. в описании симулятора.)

3.1. Динамическая маршрутизация

Динамическая маршрутизация позволяет избежать фатальных потерь пакетов и разрыва соединений.

В простейшем случае, задание динамической модели маршрутизации можно осуществить добавлением в скрипт следующей команды.

```
set ns [new Simulator]

$ns rtproto DV
…
```

Эта команда определяет протокол маршрутизации, который будет использоваться на узлах топологии сети, определенных в списке параметров команды (в данном случае команда применена ко всем узлам сети).

В общем виде команда имеет следующий формат:

```
$ns rtproto proto node-list,
```

где node-list определяет список узлов, на которых задействован маршрутизирующий протокол proto (по умолчанию определяет все узлы сети).

Добавьте описанную команду в скрипт (создав новый файл work3_2.tcl) и вновь запустите симулятор. Сразу после начала моделирования в сети начнут двигаться небольшие пакеты, отображаемые черным цветом. Щелкнув по ним можно получить информацию о том, что они являются пакетами маршрутизирующего протокола. С их помощью осуществляется сбор информации о топологии и вычисление новых маршрутов трафика. Проконтролируйте с помощью утилиты nam и trace-файла, что в результате использования маршрутизирующего протокола, в данном сценарии, при отказе линии связи исходного маршрута, движение пакетов осуществляется по другому маршруту.

3.2. Задание весовых коэффициентов (цены) соединений

Для моделирования различных способов выбора маршрута передачи, в симуляторе предусмотрена возможность задания различным звеньям сети соответствующих весовых коэффициентов (цены).

По умолчанию, все звенья сети имеют цену, равную 1. Метрикой для вычисления кратчайшего маршрута является суммарная цена всех звеньев маршрута. Понизить приоритет какого-либо звена, при определении маршрута, можно, изменив его ценовой

коэффициент (увеличив его). Для этого используются команды типа

$ns cost $*node1* $*node2 costval,*

где *costval* определяет ценовой коэффициент заданного соединения.
Модифицируйте скрипт файла work3_2.tcl таким образом, чтобы трафик двигался через узлы r4 и r5 изначально (для этого увеличьте 'цену' канала между узлами r1 и r2 (1-ый вариант задания), r2 и r3 (2-ой и 3-ий варианты) для обоих направлений, а также замените тип протокола маршрутизации с DV на Session для постоянного перерасчета маршрутов). Скрипт сохраните в файле work3_3.tcl. Проконтролируйте маршрут трафика с помощью nam.

3.3 Управление динамикой маршрутизации

Создайте скрипт, в котором цены звеньев сети изменяются в процессе моделирования. Используйте топологию, описанную в пункте 3.1 без разрыва соединения. Добавьте команды управления трафиком, такие, чтобы в момент времени 2.0 секунды, после начала моделирования, поток данных переключился с кратчайшего маршрута (начальный маршрут), на более длинный маршрут (описан в предыдущем пункте), а в момент времени 4.0 секунды после начала моделирования - снова на кратчайший маршрут (используйте протокол Static или Session и команду "$ns compute-routes" после каждого изменения цен соединений для обновления информации о топологии). Сохраните скрипт в файле work3_4.tcl.

Программа работы

1. Создайте файл work3_0.tcl с приведенным в пункте 2.1 текстом скрипта и запустите его на моделирование. Обратите внимание на команды, моделирующие отказ и восстановление канала связи.
2. Создайте скрипт в соответствии с индивидуальным заданием (п. 2.2), сохраните его в виде файла work3_1.tcl и запустите скрипт на моделирование. Обратите внимание на состояние трафика во время и после разрыва соединения. Проанализируйте результаты.
3. Модифицируйте скрипт предыдущего задания в соответствии с п..3.1, сохраните его в виде файла work3_2.tcl и запустите его на моделирование. Убедитесь, что, при использовании динамической маршрутизации, разрыв одного из звеньев исходного маршрута приводит не к разрыву логического соединения, а лишь к изменению маршрута.
4. Измените скрипт в соответствии с п. 3.2. и сохраните результаты в файле work3_3.tcl. После запуска скрипта, проконтролируйте маршрут движения трафика в соответствии с заданными весовыми коэффициентами.
5. Создайте скрипт в соответствии с п. 3.3. и сохраните результаты в файле work3_4.tcl. После запуска скрипта, проконтролируйте маршрут движения трафика в соответствии с заданными временными характеристиками.

Работа № 4
Графическое отображение результатов моделирования.
Утилита Xgraph

1. Цель работы

- ознакомление с работой утилиты графического отображения данных Xgraph;
- ознакомление с методами мониторинга трафика в симуляторе ns2.

2. Порядок работы с утилитой Xgraph

Для визуального отображения результатов моделирования кроме пакета nam часто используется сервисная программа Xgraph. Данная утилита предназначена для построения графиков в среде X-Windows. Утилита Xgraph может читать данные из входных файлов или непосредственно со стандартного ввода.

Использование данной утилиты для вывода результатов моделирования в симуляторе ns2 несколько сложнее, чем пакета nam, так как требует введения дополнительных команд в скрипт, для формирования выходного файла соответствующего формата.

При подготовке к работе следует ознакомиться с описанием утилиты Xgraph, приведенном в главе 3.

Подготовка исходных данных и запуск Xgraph

Входные данные для Xgraph задаются в виде текстового файла или стандартного потока ввода. Формат исходных данных подробно рассмотрен при описании утилиты Xgraph. Здесь лишь напомним, что входные наборы данных состоят из упорядоченного списка точек в форме

"{directive} X Y",

где 'directive' принимает значения "draw" или "move". В первом случае строится линия между предыдущей и текущей точками При выборе команды "move" линии между точками не строятся. Если команды "draw" или "move" опущены, то, по умолчанию, используется "draw".

Запуск утилиты Xgraph осуществляется командой

xgraph [*options*] <*file1*> ... <*fileN*>,

где: *options* – опции (настройки), управляющие выводом результатов.
<*file1*> ... <*fileN*> - один или несколько файлов исходных данных.

Опции вывода могут быть указаны в командной строке, в файле пользователя XWindows - .Xdefaults или .Xresources, либо в самом входном файле в виде "<option>: <value>". Опции и значения должны быть обязательно разделены пробелом.

Перечень основных опций программы Xgraph приведен в разделе 3.4.

.

3 Использования Xgraph для отображения результатов моделирования

3.1. Пример мониторинга трафика

Рассмотрим сеть показанную на рис 4.5.

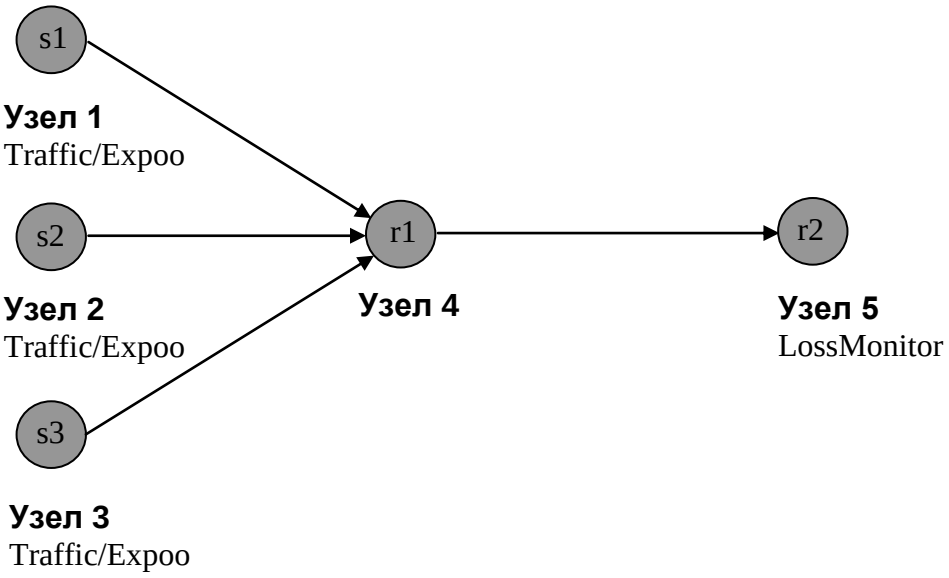


Рис. 4.5.

Сеть состоит из пяти узлов: источников s1, s2 и s3, маршрутизирующего узла r1 и приемника r2. На каждом из узлов s1 - s3 находятся источники трафика с экспоненциальным распределением on/off интервалов, прикрепленные к UDP-агентам. К узлу r2 прикреплен монитор LossMonitor используемый в качестве приемника трафика и собирающий информацию о принятых пакетах данных.

Источники трафика имеют следующие параметры:

- размер пакетов - 200 байт;
- времена "burst" и "idle", характеризующие экспоненциальное распределение, - 2 и 1 сек, соответственно;
- максимальная скорость генерации трафика - 100, 200 и 300 Кбит/сек для каждого из источников, соответственно.

Линии связи имеют полосу пропускания 1Mb и задержку распространения 100 миллисекунд. Источники трафика начинают работу через 10 сек после начала моделирования и продолжают работу 40 сек. Моделирование продолжается 60 сек.

Для исследования процессов в данной сети будем собирать информацию о трафике каждого из источников с помощью агента LossMonitor и записывать ее в соответствующие файлы. Результат мониторинга будем выводить в виде графика с помощью утилиты xgraph.

Текст скрипта

```
set ns [new Simulator]

set f0 [open out0.tr w]
set f1 [open out1.tr w]
set f2 [open out2.tr w]

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
```

```
set n3 [$ns node]
set n4 [$ns node]

$ns duplex-link $n0 $n3 1Mb 100ms DropTail
$ns duplex-link $n1 $n3 1Mb 100ms DropTail
$ns duplex-link $n2 $n3 1Mb 100ms DropTail
$ns duplex-link $n3 $n4 1Mb 100ms DropTail

proc finish {} {
    global f0 f1 f2
    close $f0
    close $f1
    close $f2
    exec xgraph out0.tr out1.tr out2.tr -geometry 800x600 \
    -0 source0 -1 source1 -2 source2 &
    exit 0
}

proc attach-expoo-traffic { node sink size burst idle rate } {
    set ns [Simulator instance]
    set source [new Agent/CBR/UDP]
    $ns attach-agent $node $source
    set traffic [new Traffic/Expoo]
    $traffic set packet-size $size
    $traffic set burst-time $burst
    $traffic set idle-time $idle
    $traffic set rate $rate
    $source attach-traffic $traffic
    $ns connect $source $sink
    return $source
}

proc record {} {
      global sink0 sink1 sink2 f0 f1 f2
      set ns [Simulator instance]
      set time 0.5
      set bw0 [$sink0 set bytes_]
      set bw1 [$sink1 set bytes_]
      set bw2 [$sink2 set bytes_]
      set now [$ns now]
      puts $f0 "$now [expr $bw0/$time*8/1000000]"
      puts $f1 "$now [expr $bw1/$time*8/1000000]"
      puts $f2 "$now [expr $bw2/$time*8/1000000]"
      $sink0 set bytes_ 0
      $sink1 set bytes_ 0
      $sink2 set bytes_ 0
      $ns at [expr $now+$time] "record"
}

set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]
```

```
$ns attach-agent $n4 $sink0
$ns attach-agent $n4 $sink1
$ns attach-agent $n4 $sink2

set source0 [attach-expoo-traffic $n0 $sink0 200 2s 1s 100k]
set source1 [attach-expoo-traffic $n1 $sink1 200 2s 1s 200k]
set source2 [attach-expoo-traffic $n2 $sink2 200 2s 1s 300k]

$ns at 0.0 "record"
$ns at 10.0 "$source0 start"
$ns at 10.0 "$source1 start"
$ns at 10.0 "$source2 start"

$ns at 50.0 "$source0 stop"
$ns at 50.0 "$source1 stop"
$ns at 50.0 "$source2 stop"

$ns at 60.0 "finish"

$ns run
```

В данном скрипте создание и прикрепление источников к узлу оформлено в виде процедуры

```
proc attach-expoo-traffic { node sink size burst idle rate },
```

параметрами которой являются ссылки на соответствующие узлы, приемники, и характеристики источников трафика.

Для сбора информации также используется отдельная процедура

```
proc record {},
```

запускаемая в начале моделирования и вызываемая через заданные интервалы времени. С ее помощью осуществляется вычисление текущих значений трафика отдельных источников и запись их в соответствующие выходные файлы. out0.tr, out1.tr и out2.tr. Эти файлы затем используются в качестве входных для утилиты Xgraph.

Запуск Xgraph осуществляется из процедуры proc finish{}. При этом, в качестве параметров запуска, используются опции указания начального размера окна Xgraph и задания имени наборов данных.

3.2 Задание к самостоятельной работе

Создайте следующую модель сети, используя топологию заданного варианта.

В узлах s1 и s2 находятся источники трафика с экспоненциальным распределением on/off интервалов, прикрепленные к UDP-агентам. К узлам k1 и k2 прикреплены приемники трафика (Null - агенты).

Источники трафика имеют следующие параметры:

- размер пакетов - 300 байт;
- времена "burst" и "idle", характеризующие экспоненциальное распределение, равны 0.1 сек;
- максимальная скорость генерации трафика – 150 Кбит/сек и 250 Кбит/сек для каждого из источников, соответственно.

Линии связи между узлами r1-r2 s1-r1 и s2-r1 имеют пропускную способность 128 Кбит/сек и задержку распространения 20 мсек; параметры остальных линий связи,

соответственно, 1Мб/сек 100 мсек. Источники трафика начинают работу через 0.1 сек, после начала моделирования и продолжают работу до момента 5.0 сек. Моделирование продолжается 6.0 сек.

В скрипте будем осуществлять мониторинг очереди соединения между узлами r1 и r2. Требуется отобразить результаты мониторинга в виде графика с помощью утилиты Xgraph.

Для осуществления мониторинга очереди можно воспользоваться соответствующим методом monitor-queue. Будем собирать информацию об изменении размера очереди. Метод monitor-queue создает объект типа QueueMonitor, имеющий ряд переменных состояния, среди которых нас будет интересовать переменная pkts_, определяющая размер очереди в пакетах. Создать данный объект можно, используя команду

set qm0 [$ns monitor-queue $node1 $node2 [$ns get-ns-traceall]],

в результате выполнения которой будет создан объект qm0 типа QueueMonitor, с помощью которого можно производить мониторинг очереди в линии между узлами node1 и node2 (подробнее см. общее описание симулятора - раздел 'Методы TRACE и MONITORING').

Установите размер очереди равным 30 пакетов.

Для считывания данных о размере очереди, можно использовать процедуру, аналогичную использованной в предыдущем примере.

```
proc trqueue {} {
    global qm0 f0
    set ns [Simulator instance]
    set time 0.1
    set ql [$qm0 set pkts_]
    set now [$ns now]
    puts $f0 "$now $ql"
    $qm0 reset
    $ns at [expr $now+$time] "trqueue"
}
```

В этой процедуре осуществляется мониторинг очереди с помощью объекта qm0. Состояние очереди контролируется через заданный интервал времени, определяемый переменной time, и осуществляется запись результатов в выходной файл, определяемый переменной f0.

При выводе результатов моделирования с помощью Xgraph в параметрах запуска укажите заголовок графика, имя набора данных, обозначения осей, а также задайте размер и начальное положение окна Xgraph. Скрипт сохраните в файле work4_1.tcl.

Кроме мониторинга размера очереди объекты типа QueueMonitor способны отслеживать число поступающих, отправляемых, а также отброшенных в очереди пакетов. Добавьте в скрипт команды мониторинга поступивших и отброшенных пакетов в очереди между узлами r1 и r2 (используйте переменные состояния монитора parrivals_ и pdrops_). Результаты мониторинга выведите в файлы out1.tr и out2.tr и отобразите с помощью Xgraph. Скрипт сохраните в файле work4_2.tcl.

4. Процедура обработки данных для Xgraph

4.1. Пример обработки данных для Xgraph

В предыдущем задании входные данные для xgraph формировались непосредственно командами симулятора. Часто удобно для дальнейшего использования, в том числе для визуального отображения, выходные данные симулятора предварительно обработать (отфильтровать, записать в определенном формате и т.д.)

Следующий скрипт иллюстрирует подобную процедуру. Будем использовать сеть с топологией предыдущего примера. Установим в ней размер пакетов источника - 500 байт, размер очереди r1-r2 - 15 пакетов, время остановки источников 2.5 сек, а время моделирования 3.0 сек.

Текст скрипта

```
set ns [new Simulator]

#Процедура обработки выходных данных.

proc finish { label mod } {

#Создаем и подготавливаем выходной файл данных.

    exec rm -f temp.rands
    set f [open temp.rands w]
    puts $f "TitleText: $label"
    puts $f "Device: Postscript"
    exec rm -f temp.p
    exec touch temp.p
```

#Обрабатываем файл данных симулятора out.tr и заносим из него данные о полученных/отправленных пакетах очереди во временный файл temp.p.

```
    exec awk {
        {
            if (($1 == "+" || $1 == "-" ) && \
               ($5 == "exp"))\
                    print $2, $8*(mod+10) + ($11 % mod)
        }
    } mod=$mod out.tr > temp.p
```

#Заносим данные об отброшенных пакетах очереди во временный файл temp.p.

```
    exec rm -f temp.d
    exec touch temp.d
    exec awk {
        {
            if ($1 == "d")
                print $2, $8*(mod+10) + ($11 % mod)
        }
    } mod=$mod out.tr > temp.d
```

#Заносим данные из временных файлов temp.p и temp.d в выходной файл для xgraph temp.rands.

```
        puts $f \"enque/deque
        flush $f
        exec cat temp.p >@ $f
        flush $f

        puts $f \n\"drops
        flush $f

        exec head -1 temp.d >@ $f
        exec cat temp.d >@ $f
        close $f
        set tx "time (sec)"
        set ty "packet number (mod $mod)"

    #Запускаем Xgraph со входным файлом temp.rands.

        exec xgraph -bb -tk -nl -m -zg 0 -x $tx -y $ty temp.rands &
        exit 0
    }
```

#Процедура создания и прикрепления к узлу источника трафика с экспоненциальным распределением on/off интервалов.

```
    proc attach-expoo-traffic { node sink size burst idle rate } {
        set ns [Simulator instance]
        set source [new Agent/CBR/UDP]
        $ns attach-agent $node $source
        set traffic [new Traffic/Expoo]
        $traffic set packet-size $size
        $traffic set burst-time $burst
        $traffic set idle-time $idle
        $traffic set rate $rate
        $source attach-traffic $traffic
        $ns connect $source $sink
        return $source
    }

    #Задаем заголовок графика и период вывода номеров пакетов.

    set label "Expoo_Trffic"
    set mod 50

    $ns color 0 Blue
    $ns color 1 Red

    #Создаем топологию сети.

    set n0 [$ns node]
    set n1 [$ns node]
    set n2 [$ns node]
    set n3 [$ns node]
```

```
$ns duplex-link $n0 $n2 1Mb 100ms DropTail
$ns duplex-link $n1 $n2 1Mb 100ms DropTail
$ns duplex-link $n2 $n3 128kb 100ms DropTail

$ns queue-limit $n2 $n3 10

#Подготавливаем выходной файл симулятора out.tr.
exec rm -f out.tr
set fout [open out.tr w]

#Задаем мониторинг очереди.

$ns trace-queue $n2 $n3 $fout

set sink0 [new Agent/Null]
set sink1 [new Agent/Null]
$ns attach-agent $n3 $sink0
$ns attach-agent $n3 $sink1

$ns queue-limit $n2 $n3 15
```

#Создаем источники трафика и задаем их идентификаторы, используемые для различения пакетов разных источников.

```
set source0 [attach-expoo-traffic $n0 $sink0 500 0.1s 0.1s 150k]
$source0 set fid_ 0
set source1 [attach-expoo-traffic $n1 $sink1 500 0.1s 0.1s 250k]
$source1 set fid_ 1

#Задаем временную последовательность внешних событий.

$ns at 0.1 "$source0 start"
$ns at 0.1 "$source1 start"

$ns at 2.5 "$source0 stop"
$ns at 2.5 "$source1 stop"

#Закрываем выходной файл и запускаем процедуру finish.

$ns at 3.0 "ns flush-trace; close $fout; \
            finish $label $mod"

$ns run
```

При выполнении данного скрипта будет создан trace файл out.tr стандартного формата симулятора с данными мониторинга очереди соединения между узлами r1 и r2. Процедура finish создает вспомогательные файлы temp.p и temp.d с информацией, соответственно, об успешно переданных и отброшенных пакетах в очереди рассматриваемого соединения.

Далее эти данные объединяются и заносятся в соответствующем формате во входной файл для xgraph - temp.rands. Таким образом, во входном файле имеется два набора данных (полученные/отправленные очередью пакеты данных и отброшенные пакеты.

Для удобства вывода, номера пакетов выводятся с периодом 50 (mod 50). Пакеты источников для возможности их различения отображаются с некоторым сдвигом: пакеты 1-го источника отображаются с номерами с 0 по 50, а 2-го источника - с 60 по 110 (т.е. к номерам пакетов добавлено 60). В окне xgraph по оси X отложено время моделирования, по оси Y - номера пакетов.

4.2. Задание к самостоятельной работе

Создайте и прикрепите к узлам s1, s2 и s3 источники трафика с экспоненциальным распределением on/off интервалов.

Источники соедините с приемниками трафика на узлах k1, k2 и k3, соответственно.

Установите следующие параметры источников:

- размер пакетов источников: s1 и s2 - 500 байт, s3 - 1000 байт;
- скорость генерации трафика: s1 – 150 Кбит/сек, s2 - 250 Кбит/сек, s3 - 100 Кбит/сек;
- времена "burst" и "idle" - по 0.1 сек;
- размер очереди r1-r2 - 15 пакетов.

Время запуска источников - 0.1 сек, время остановки - 2.5 сек, время моделирования - 3.0 сек.

Используя процедуру finish предыдущего примера, произведите мониторинг очереди соединения r1-r2. Не забудьте указать идентификаторы пакетов источников для корректного отображения данных в Xgraph. Скрипт сохраните в файле work4_4.tcl.

5. Программа работы

3. Ознакомьтесь с описанием утилиты Xgraph, форматом ее входных данных и пользовательским интерфейсом.
4. Создайте файл work4_0.tcl с приведенным в пункте 3.1 текстом скрипта и запустите его на моделирование. Ознакомьтесь с особенностями работы пакета xgraph.
5. В соответствии с заданием пункта 3.2 произведите моделирование работы сети с заданной топологией. Скрипт сохраните в файле work4_1.tcl.
6. Произведите модификацию топологии в соответствии с заданием пункта 3.2, результат сохраните в файле work4_2.tcl.
7. Создайте скрипт содержащий команды обработки выходного файла симулятора пункта 4.1 (файл work4_3.tcl), ознакомьтесь с приведенными в примере командами.
8. Создайте скрипт в соответствии с заданием пункта 4.2., сохраните его в виде файла work4_4.tcl, и запустите на моделирование. Проконтролируйте корректное отображение данных всех трех источников трафика.

Работа №5 Исследование алгоритмов протокола TCP

1. Цель работы

- приобретение навыков исследования протоколов различного уровня с помощью симулятора ns2
- ознакомление с особенностями протокола транспортного уровня - TCP;
- исследование алгоритмов протокола TCP версии Tahoe.

2. Исследование протоколов транспортного уровня

Многоуровневая архитектура взаимодействия объектов в сетях связи допускает возможность совершенствования протоколов различных уровней. При отладке новых версий протоколов широко применяется моделирование с использованием симулятора ns2.

Рассмотрим простейшую модель, позволяющую исследовать эффективность конкретной версии протокола транспортного уровня – TCP.

Для отображения результатов моделирования будем применять утилиту Xgraph, используя навыки, приобретенные в работе N 4.

2.1. Пример модели сети для анализа протокола TCP

Простейшая модель сети для анализа протокола TCP показана на рис. 4.6.

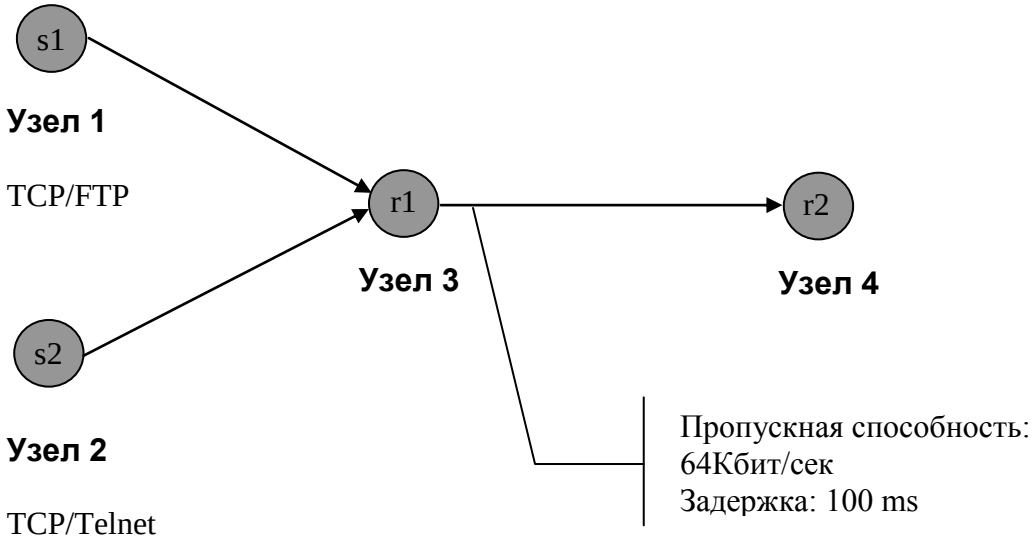


Рис. 4.6.

Сеть состоит из четырех узлов: источников s1 и s2, коммугирующего узла r1 и приемника r2, и линий связи между ними. В узле s1 находится источник ftp трафика, прикрепленный к агенту TCP, в узле s2 - источник telnet трафика, также использующий транспортный протокол TCP.

Параметры TCP для обоих узлов:

- размер пакетов 100 байт,
- максимальный размер окна переполнения - 10 пакетов.

Параметры FTP источника: число генерируемых пакетов - 200. Параметры источника telnet трафика: средний интервал между генерируемыми пакетами - 0.02 сек. К узлу r1 не прикреплено никаких источников, в узле r2 находится приемник трафика.

Все линии связи имеют задержку распространения 100 миллисекунд.

Пропускная способность линий s1-r1 и s2-r1 составляет 1Мбит/сек.

Пропускная способность линии r1-r2 - 64Кбит/сек.

Размер очереди соединения между узлами r1 и r2 - 5 пакетов. Источник ftp-трафика запускается через 0.1 сек после начала моделирования и прекращает работу после генерации 200 пакетов, источник telnet трафика запускается и останавливается через 0.5 и 1.5 сек, соответственно. Моделирование продолжается 6.0 сек.

Для исследования процессов в данной сети будем осуществлять мониторинг очереди соединения между узлами r1 и r2. Результаты будем выводить на экран в виде графика с помощью утилиты Xgraph. При этом будем отображать зависимость номеров поступающих и покидающих данную очередь пакетов от времени. Для этого модифицируем процедуру finish, добавив в нее команды вывода информации в файл out.rands, в формате, используемом в Xgraph (см. описание утилиты Xgraph).

Текст скрипта

```
set ns [new Simulator]

proc finish { file mod } {

# Подготавливаем файл temp.rands для вывода результатов с помощью Xgraph.
        exec rm -f temp.rands
        set f [open temp.rands w]
        puts $f "TitleText: $file"
        puts $f "Device: Postscript"

# Обрабатываем файл результатов моделирования out.tr.
# Выводим информацию о передаваемых пакетах во временный файл temp.p.

        exec rm -f temp.p
        exec touch temp.p

        exec awk {
                {
                        if (($1 == "+" || $1 == "-" ) && \
                           ($5 == "tcp" || $5 == "ack"))\
                                        print $2, ($8-1)*(mod+10) + ($11 % mod)
                }
        } mod=$mod out.tr > temp.p

# Выводим информацию об отброшенных пакетах во временный файл temp.d.

        exec rm -f temp.d
        exec touch temp.d
        exec awk {
                {
                        if ($1 == "d")
                                print $2, ($8-1)*(mod+10) + ($11 % mod)
```

```
            }
        } mod=$mod out.tr > temp.d

# Обрабатываем файл результатов моделирования out2.tr.  Выводим
# информацию о пакетах подтверждений (ACK)  во временный файл temp.p2.

        exec rm -f temp.p2
        exec touch temp.p2
        exec awk {
            {
                if (($1 == "-" ) && \
                   ($5 == "tcp" || $5 == "ack"))\
                        print $2, ($8-1)*(mod+10) + ($11 % mod)
            }
         } mod=$mod out2.tr > temp.p2

# Объединяем информацию с соответствующими заголовками из файлов
# temp.p, temp.d и temp.p2 во входной файл для утилиты Xgraph  - temp.rands.

        puts $f \"packets
        flush $f
        exec cat temp.p >@ $f
        flush $f
        puts $f \n\"acks
        flush $f
        exec cat temp.p2 >@ $f

        puts $f [format "\n\"skip-1\n0 1\n\n"]

        puts $f \"drops
        flush $f
        exec head -1 temp.d >@ $f
        exec cat temp.d >@ $f
        close $f
        set tx "time (sec)"
        set ty "packet number (mod $mod)"

    # Запускаем Xgraph.

        exec xgraph -bb -tk -nl -m -zg 0 -x $tx -y $ty temp.rands &
        exit 0
    }

    # Задаем заголовок графика и период вывода номеров пакетов.

    set label "tcp/ftp+telnet"
    set mod 80

    # Задаем топологию сети.

    $ns color 1 Blue
    $ns color 2 Red
```

```
set s1 [$ns node]
set s2 [$ns node]
set r1 [$ns node]
set r2 [$ns node]

$ns duplex-link $s1 $r1 1Mb 50ms DropTail
$ns duplex-link $s2 $r1 1Mb 50ms DropTail
$ns duplex-link $r1 $r2 64kb 100ms DropTail

$ns duplex-link-op $s1 $r1 orient right-down
$ns duplex-link-op $s2 $r1 orient right-up
$ns duplex-link-op $r1 $r2 orient right

$ns queue-limit $r1 $r2 5

$ns duplex-link-op $r1 $r2 queuePos 0.5

#Подготавливаем файлы out.tr и out2.tr для вывода результатов.

exec rm -f out.tr
set fout [open out.tr w]
$ns trace-queue $r1 $r2 $fout
exec rm -f out2.tr
set fout2 [open out2.tr w]
$ns trace-queue $r2 $r1 $fout2

# Создаем источники и приемники трафика.

set snk1 [new Agent/TCPSink]
$ns attach-agent $r2 $snk1

set snk2 [new Agent/TCPSink]
$ns attach-agent $r2 $snk2

set tcp1 [new Agent/TCP]
$tcp1 set maxcwnd_ 12
$tcp1 set packetSize_ 100
$ns attach-agent $s1 $tcp1
$ns connect $tcp1 $snk1
$tcp1 set fid_ 1
set ftp1 [$tcp1 attach-source FTP]

set tcp2 [new Agent/TCP]
$tcp2 set maxcwnd_ 12
$tcp2 set packetSize_ 100
$ns attach-agent $s2 $tcp2
$ns connect $tcp2 $snk2
$tcp2 set fid_ 2
set tln1 [$tcp2 attach-source Telnet]
$tln1 set interval_ 0.02s
```

```
# Задаем хронологию событий.

$ns at 0.1 "$ftp1 produce 200"
$ns at 0.5 "$tln1 start"

$ns at 1.5 "$tln1 stop"

# Закрываем выходные файлы и вызываем процедуру обработки и вывода
# результатов finish.

$ns at 6.0 "ns flush-trace;  \
        close $fout; close $fout2; \
        finish $label $mod"
$ns run
```

При выполнении данного скрипта будут созданы trace файлы out.tr и out2.tr стандартного формата симулятора с результатами мониторинга очередей соединения между узлами r1 и r2 (соответственно r1-r2 и r2-r1). Процедура finish создает вспомогательные файлы temp.p, temp.d и temp.p2 с информацией, соответственно, об успешно переданных, отброшенных и подтверждающих (ACK) пакетах в рассматриваемом соединении.

Далее эти наборы данных объединяются и заносятся с соответствующими заголовками во входной файл для Xgraph - temp.rands. После выполнения скрипта, во входном файле будет находиться четыре набора данных (полученные/отправленные очередью пакеты данных, отправленные подтверждения на эти пакеты, а также отброшенные пакеты. Для отображения отброшенных пакетов маркерами в форме крестов (ими маркируется четвертый набор данных в xgraph), перед набором данных отброшенных пакетов добавлен пустой набор данных с меткой skip-1).

После формирования выходных данных запускается сама утилита Xgraph с рядом параметров (параметры запуска и интерфейс приведен в описании к Xgraph). Для удобства вывода, номера пакетов выводятся с периодом 80 (mod 80). Пакеты разных источников для возможности различения отображаются с некоторым сдвигом: пакеты ftp источника отображаются с номерами с 0 по 80, а telnet источника - с 90 по 170. В окне Xgraph по оси X отложено время моделирования, по оси Y - номера пакетов.

2.2. Задание к работе

Создайте модель сети, используя топологию заданного варианта, со следующими параметрами:

В узле s1 находится источник FTP трафика, прикрепленный к агенту TCP.

Параметры протокола TCP (TCP Tahoe):

- размер пакетов 100 байт;
- максимальный размер окна переполнения - 15 пакетов.

Параметры FTP источника:

- число генерируемых пакетов - 200.

В узле k2 находится приемник трафика TCP/Sink.

Линия связи между узлами r1 и -r2 имеет пропускную способность 256 Кбит/сек и задержку распространения 200 миллисекунд. (Для варианта N2 параметры линии связи r1-r2, соответственно, 200 Кбит/сек и 200 миллисекунд.)

Все остальные каналы связи имеет пропускную способность 2Мбит/сек и задержку распространения 10 миллисекунд,

Размер очереди соединения r1-r2 - 8 пакетов. Источник ftp трафика запускается через 0.1 сек после начала моделирования и прекращает работу после генерации 200 пакетов. Моделирование продолжается 6.0 сек.

В скрипте предусмотрите мониторинг очереди линии связи между узлами r1 и r2, отобразите изменение состояния очереди с помощью утилиты Xgraph.

Для этого можно использовать вариант процедуры finish, применявшийся в предыдущем примере. Скрипт сохраните в файле work5_1.tcl.

2.3. Расшифровка результатов моделирования.

(Алгоритмы управления передачей на различных фазах протокола TCP)

После выполнения скрипта work5_1.tcl результаты моделирования будут выведены в окне xgraph. Сохраните его копию в файле xg5_1.ps формата Postscript (см. описание xgraph). На полученном графике можно проследить ряд применяемых в протоколе TCP Tahoe алгоритмов:

Slow Start (медленный старт),

Fast Retransmit (быстрый повтор передачи) и

Congestion Avoidance (предупреждение перегруженности).

На графике видно, что пакеты с 0-го по 29-ый переданы без потерь, при этом, окно TCP экспоненциально возрастает с 1 до 15 в соответствии с алгоритмом медленного старта. Каждый пакет данных отображается с помощью маркера в форме квадрата, в моменты времени, когда он поступает и покидает очередь, соответственно. Первые пакеты практически не задерживаются в очереди и метки, отображающие их движение, на графике сливаются. С увеличением загруженности очереди время между поступлением в нее пакета и его отправлением начинает увеличиваться, что видно по увеличению расстояния на графике между метками. Во время прибытия 30-го пакета очередь полностью заполнена, и пакет, в соответствии с методом обработки очереди DropTail, отбрасывается.

После получения подтверждения доставки (ACK) 29-го пакета, источник получает ряд дублирующих подтверждений для 29-го пакета, вызванных успешной доставкой пакетов следующего окна данных. После получения третьего дублирующего подтверждения (четвертое подтверждение для 29-го пакета) источник переходит в режим быстрого повтора передачи и медленного старта. Источник уменьшает “окно перегруженности” TCP до одного пакета и повторяет передачу 30-го пакета. Дополнительно, порог медленного старта sstresh (максимальное значение окна перегруженности) уменьшается до 7 (0.5 max[min(window_,cwnd_),2]).

Приемник, уже имея пакеты следующего окна данных, получив 30-ый пакет в результате повторной передачи, посылает подтверждение последнего пакета следующего сегмента. После получения подтверждения на этот пакет источник увеличивает окно перегруженности на 1 и продолжает передачу. Достигнув порога медленного старта, источник переходит в режим устранения перегруженности. Далее, окно отправителя увеличивается, как и предполагалось, на один пакет за каждый цикл RTT (round trip time).

Создайте скрипт work5_2.tcl для моделирования ситуации, при которой происходит потеря трех пакетов в окне данных. Этого можно добиться, изменяя в предыдущем сценарии величину максимального размера очереди и/или максимального окна переполнения.

3. Мониторинг состояния протокола

3.1. Процедура сбора и вывода информации о текущем состоянии протокола.

Проследить изменения параметров протокола (таких, как размер окна переполнения, граница медленного старта и т.д.) можно, добавив в текст скрипта ряд команд для сбора и вывода информации о состоянии TCP-соединения. Эти TcL-команды удобно оформить в виде рекурсивной процедуры, периодически вызываемой через заданный интервал времени.

В приведенном ниже тексте процедуры dump осуществляется вывод текущих значений времени моделирования (time), последовательного номера передаваемого пакета (t_seqno), окна переполнения (cwnd), границы медленного старта (ssthresh) и наивысшего значения из полученных подтверждений от приемника (ack). Полный список переменных состояния протокола приведен в описании симулятора.

```
proc dump { src interval } {
    global ns tf
    puts $tf [format "time=%5.2f; t_seqno=%3.0f; cwnd=%3.0f; ssthresh=%3.0f; \
    ack=%d;" [$ns now] [$src set t_seqno_] [$src set cwnd_] \
    [$src set ssthresh_] [$src set ack_]]
    $ns at [expr [$ns now] + $interval] "dump $src $interval"
}
```

Процедура dump имеет два параметра 'src' и 'interval'. Параметр 'src' задает TCP-объект, за которым осуществляется наблюдение, а параметр 'interval' определяет период сбора информации. Данная процедура может быть помещена в начале скрипта вместе с процедурой 'finish', а ее вызов может быть осуществлен следующим образом:

```
set tf [open time.tr w]
$ns at 0.0 "dump $tcp1 0.01"
```

При вызове процедуры осуществляется сбор информации с периодом, определяемым переменной 'interval' (в данном случае 0.01 сек) в файл time.tr.

3.2. Задание к работе

Добавьте в сценарий моделирования work5_2.tcl процедуру сбора информации и ее вызов. Скрипт сохраните в файле work5_3.tcl. Запустив симулятор с данным сценарием, получите выходной файл данных time.tr.

4. Программа работы

1. Создайте файл work5_0.tcl с приведенным в пункте 2.1 текстом скрипта и запустите его на моделирование. Ознакомьтесь с особенностями новых команд скрипта.
2. Создайте скрипт в соответствии с индивидуальным заданием (п. 2.2), сохраните его в виде файла work5_1.tcl и запустите на моделирование. Сохраните результаты моделирования в файле xg5_1.ps формата Postscript (см. описание xgraph).
3. На полученном графике (xg5_1.ps) проследите выполнение различных алгоритмов работы протокола TCP, пользуясь описанием п.2.3.
4. Модифицируйте скрипт в соответствии с заданием п. 2.3., сохраните его в файле work5_2.tcl, и запустите на моделирование. Отметьте различия в работе протокола TCP в данном и предыдущем случае.

5. Модифицируйте скрипт work5_2.tcl в соответствии с заданием п.3.2. Сохраните его в виде файла work5_3.tcl и запустите на моделирование. Проследите работу протокола на основе данных выходного файла time.tr.
6. В отчете приведите распечатку графика xg5_1.ps и отметьте границы различных фаз протокола TCP.

Работа № 6
Моделирование локальных вычислительных сетей (Ethernet)

1. Цель работы

- ознакомление с возможностями симулятора по моделированию локальных вычислительных сетей;
- изучение структуры интерфейса локальных сетей (Ethernet) в симуляторе ns2;
- ознакомление с методами многоабонентной (multicast) передачи в локальных сетях.

2. Особенности моделирования локальных вычислительных сетей

Функционирование локальных вычислительных сетей (Local-Area-Network - LAN), в том числе, беспроводных (wireless), отличается от работы магистральных сетей с соединениями точка-точка. В первую очередь, это отличие заключается в наличии общего, для всех узлов LAN, канала связи и необходимости арбитража, при доступе к общему ресурсу.

Для моделирования LAN в симуляторе NS2 используется метод make-lan, и специальный объект LanNode, моделирующий использование общего ресурса.

Общее описание процедуры make-lan имеет вид:

Simulator instproc make-lan {nodes bw delay lltype ifqtype mactype chantype}

В отличие от метода создания линий связи точка-точка (duplex-link), метод make-lan использует в качестве параметров список узлов LAN (nodes), пропускную способность(bw), задержку (delay), а также типы объектов канального уровня (lltype), интерфейса очереди (ifqtype), подуровня доступа к среде (mactype) и физического уровня (chantype).

Ниже представлен пример создания локальной сети из 2 узлов с простейшей реализацией канального уровня (LL), интерфейсом очереди DropTail, методом доступа CSMA/CD (Carrier-Sense Multiple-Access/Collision Detection - коллективный доступ к прослушиванию несущей с выявлением коллизий) и простейшим типом физического уровня (Channel).

$ns make-lan "$n1 $n2" $bw $delay LL Queue/DropTail Mac/Csma/Cd Channel

Архитектура LAN включает нижние уровни эталонной модели взаимодействия открытых систем ISO/OSI.:

- Link Layer (LL)
- Medium Access Control (MAC)
- Physical (PHY) Layer .

Для адекватного моделирования процессов на нижних (внутрисетевых) уровнях взаимодействия в симуляторе предусмотрены дополнительные классы объектов, в соответствии с рис. 4.7..

При моделировании передачи данных, пакет продвигается вниз по стеку протоколов, проходит через уровень управления звеном передачи (LL), включающий буфер (Queue), подуровень управления доступом (MAC) и физический уровень (Channel). Далее он поступает к объекту Classifier/Mac и, поднимаясь по стеку к вышележащим уровням, проходит через уровни Mac и LL.

В симуляторе физический уровень моделируется с помощью двух отдельных объектов Channel и Classifier/Mac. Объект Channel моделирует распределенную среду и поддерживает механизм контроля доступа MAC-объектов со стороны отправителя данных. Со стороны получателя объекты Classifier/Mac отвечают за доставку пакетов к объектам MAC-уровня.

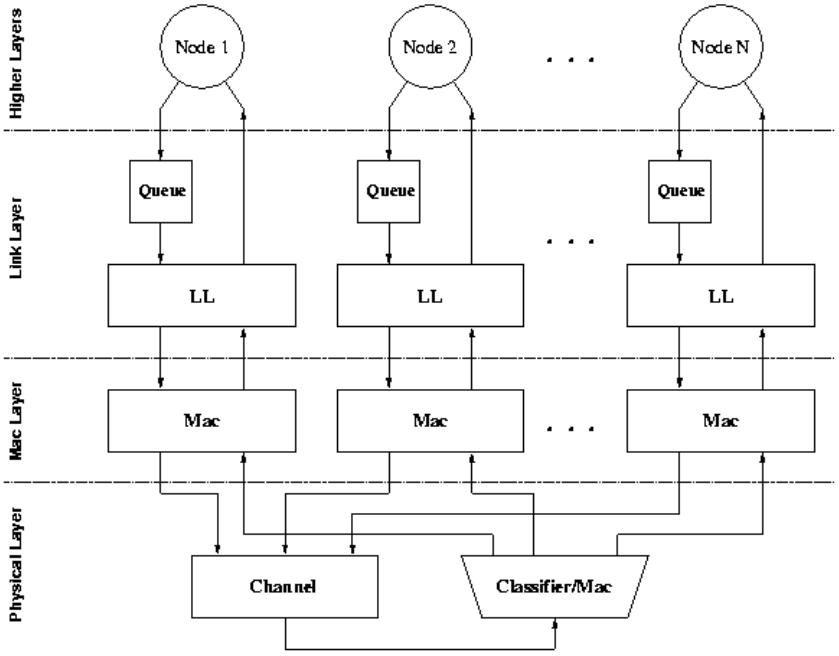


Рис. 4.7.

Класс Channel

Класс Channel используется для моделирования передачи пакета на физическом уровне. Простейшие объекты этого класса реализуют распределенную среду с поддержкой механизмов моделирования конфликтов. Это позволяет объектам Mac-уровня использовать такие функции, как прослушивание несущей, а также обнаружение и обработку коллизий. Если в канале происходит одновременная передача нескольких источников, то объект channel устанавливает флаг коллизии. С помощью проверки состояния этого флага объект типа Mac может задействовать механизмы обнаружения и обработки коллизий.

Кроме этого, в функции объектов типа Channel входит доставка пакетов на вышестоящий (Mac) уровень после окончания времени передачи и задержки распространения.

Класс Mac

Объекты класса Mac моделируют работу протоколов доступа к распределенной среде нижележащего уровня, которые необходимы при работе с беспроводными и локальными сетями. Поскольку механизмы пересылки и приема тесно связаны, то используются дуплексные Mac-объекты.

На стороне отправителя Mac-объект отвечает за добавление Mac-заголовка и транспортировку пакета в канал. При отправке MAC-объект должен использовать определенный протокол доступа к среде, прежде чем передать пакет в канал. Со стороны

приемника Mac-объект асинхронно принимает пакеты с физического уровня. После обработки пакетов на Mac-уровне, пакеты пересылаются на уровень LL.

В зависимости от типа физического уровня, Mac-уровень должен содержать определенный набор функций, таких как прослушивание несущей (carrier sense), обнаружение коллизий (collision detection), предупреждение коллизий (collision avoidance) и т.д. Поскольку эти функции влияют, как на передающую, так и на принимающую сторону, то они реализованы в одном MAC объекте.

Класс LL (link-layer)

Расположенный над MAC уровнем, уровень канала связи обладает такими функциями, как обработка очереди и повторная передача на уровне канала связи. Необходимость поддержки большого числа схем уровня канала связи привела к разделению объекта на два компонента: Очередь (Queue) и собственно уровень канала связи (LL). Объект Queue моделирует интерфейс очереди и принадлежит к общему классу Queue. С помощью объекта LL реализуются соответствующие протоколы данного уровня (такие как ARQ).

Локальные сети и процессы маршрутизации

При создании модели сети с помощью команд make-lan или newLan, формируется дополнительный объект типа LanNode - так называемый, виртуальный узел сети. Он используется для поддержки маршрутизации в моделях, включающих локальные сети. Объект LanNode объединяет такие распределенные объекты сети как Channel, Classifier/Mac и LanRouter. После этого для каждого узла LAN создается объект сетевого интерфейса (LanIface object). Этот объект содержит все необходимые объекты сетевого стека: объекты очереди (Queue), канального уровня (LL), Mac и т.д. Между объектами Node и LanNode мало общего, объект LanNode является узлом, только с точки зрения маршрутизации. И в этом смысле, представляет собой просто дополнительный узел, соединенный с каждым из узлов сети (Рис. 4.8).

Соединения LanNode с другими узлами также являются 'виртуальными' (Vlink). По умолчанию, 'цена' такого соединения равняется 1/2, таким образом, совокупность двух промежуточных виртуальных соединений (n1 - LAN - n2) имеет единичную цену обычного звена передачи - хопа.

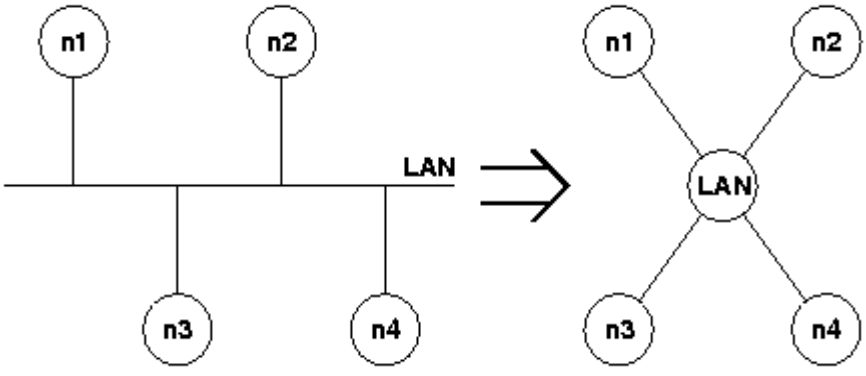


Рис. 4.8

2.1. Пример модели локальной сети

Пусть моделируемая сеть имеет вид, показанный на рис. 4.9..

Сеть состоит из пяти узлов n0 - n4, объединенных в локальную сеть и двух внешних узлов s1 и k1, соединенных с узлами локальной сети. Все линии связи имеют пропускную способность 1.5 Мбит/сек и задержку распространения 30 мсек.

Между узлами s1 и k1 работает протокол FTP, (причем, узел s1 является источником ftp-трафика, а узел k1 - приемником).

Сеанс FTP характеризуется размером передаваемого файла - 100 пакетов.

Параметры протокола TCP :

- размер пакетов 1000 байт;
- максимальный размер окна насыщения - 15 пакетов.

Моделирование продолжается 2.0 сек.

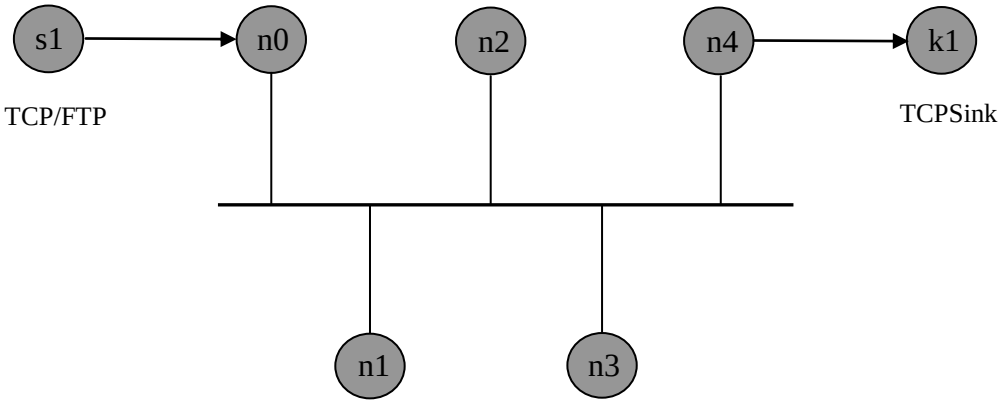


Рис. 4.9

Рассмотрим скрипт, моделирующий указанный сеанс FTP в сети заданной топологии и отображающий результат с помощью утилиты nam.

Текст скрипта

```
set opt(tr)         out.tr
set opt(namtr)      out.nam
set opt(stop)       2
set opt(node)       5

set opt(qsize)      50
set opt(bw)         1.5Mb
set opt(delay)      30ms
set opt(ll)         LL
set opt(ifq)        Queue/DropTail
set opt(mac)        Mac
set opt(chan)       Channel

proc finish {} {
    global ns opt

    $ns flush-trace
    exec nam $opt(namtr)
    exit 0
}
```

```
proc create-topology {} {
    global ns opt
    global lan n s1 k1

    set num $opt(node)
    for {set i 0} {$i < $num} {incr i} {
            set n($i) [$ns node]
            lappend nodelist $n($i)
    }

    set lan [$ns make-lan $nodelist $opt(bw) \
                    $opt(delay) $opt(ll) $opt(ifq) $opt(mac) $opt(chan)]

    set s1 [$ns node]
    $ns duplex-link $s1 $n(0) 1.5Mb 30ms DropTail
    $ns duplex-link-op $s1 $n(0) orient right
    set k1 [$ns node]
    $ns duplex-link $k1 $n(4) 1.5Mb 30ms DropTail
    $ns duplex-link-op $k1 $n(4) orient left
}

set ns [new Simulator]

set trfd [open $opt(tr) w]
$ns trace-all $trfd

set ntrf [open $opt(namtr) w]
$ns namtrace-all $ntrf

create-topology
set tcp0 [new Agent/TCP]
$ns attach-agent $s1 $tcp0
set snk0 [new Agent/TCPSink]
$ns attach-agent $k1 $snk0

$ns connect $tcp0 $snk0

$tcp0 set window_ 15

set ftp0 [$tcp0 attach-source FTP]

$ns at 0.0 "$ftp0 produce 100"
$ns at $opt(stop) "finish"
$ns run
```

В самом начале скрипта задается ряд параметров модели, для хранения которых используется массив opt. Задаются параметры интерфейса узлов локальной сети, количество узлов, время окончания моделирования и т.д. Создание топологии оформлено в виде отдельной процедуры ('create-topology').

Созданные узлы (массив n) заносятся в список nodelist командой 'lappend'. Этот список узлов используется в качестве параметра команды make-lan, создающей интерфейс локальной сети ('lan'). Остальные параметры make-lan определяют тип интерфейсов уровня

канала связи, Mac и физического уровня, а также задержку распространения и пропускную способность канала связи.

2.2. Задание к самостоятельной работе

Создайте модель сети, показанной на рис. 4.10.

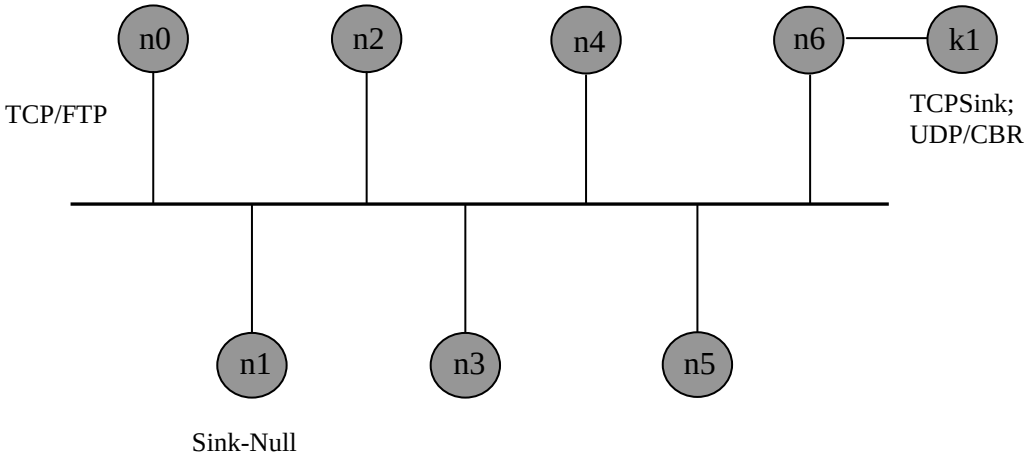


Рис. 4.10

Сеть состоит из семи узлов n0 - n6, объединенных в локальную сеть, и внешнего узла k1, соединенного с узлом n6 локальной сети. Пропускная способность канала локальной сети 10 Мбит/сек, задержка распространения между любой парой узлов - 1,0 мсек; пропускная способность канала n6-k1 составляет 8 Мбит/сек, а задержка распространения 2,0 мсек.

Между узлами n0 и k1 работает протокол FTP (узел n0 является источником ftp-трафика, а узел k1 - приемником), а между узлами k1 и n1 - протокол передачи данных, генерируемых с постоянной скоростью (CBR), использующий транспортный протокол UDP (узел k1 является источником cbr-трафика, а n1 - приемником).

Параметры протокола TCP:

- размер пакетов - 500 байт;
- максимальный размер окна насыщения - 15 пакетов.

Сеанс FTP начинается сразу после начала моделирования и характеризуется размером передаваемого файла - 100 пакетов.

Параметры CBR источника:

- размер пакетов - 500 байт;
- интервал между генерацией пакетов - 0.02 сек.

Сеанс cbr запускается через 0.1 сек после начала моделирования.

Для моделирования CBR-источника в связке с протоколом UDP можно использовать следующий набор команд:

```
set udp0 [new Agent/UDP]
$ns attach-agent $node1 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packet_size_ 500
$cbr0 set interval_ 0.02
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
```

```
$ns attach-agent $n(1) $null0
$udp0 set dst_ [$null0 set addr_]
```

Последняя команда указывает приемник для UDP-агента.

Время окончания моделирования 5.0 сек.

Результаты моделирования отобразите с помощью утилиты nam., для различения потоков используйте разные цвета пакетов.

3. Моделирование многоадресной передачи

Многоадресная передача (multicast) подразумевает передачу данных одновременно нескольким абонентам сети. Для моделирования многоадресной передачи в симуляторе ns используются методы multicast-маршрутизации. Возможность использования multicast-маршрутизации должна быть указана перед созданием топологии в виде требования к общему классу Simulator. с помощью команды

```
set ns [ new Simulator -multicast on ]
```

Стратегия multicast-маршрутизации представляет собой механизм, с помощью которого определяется дерево маршрутов пересылки данных абонентам. В симуляторе поддерживается несколько протоколов multicast маршрутизации: (Centralized Multicast, Dense Mode, Shared Tree Mode и Bi-directional Shared Tree Mode). Подробное описание этих протоколов приведено в оригинальном описании симулятора ns2.

Тип используемого протокола задается с помощью метода класса Simulator - mrtproto {}. В общем случае, метод mrtproto имеет дополнительные аргументы, определяющие список узлов, на которых будет запущен соответствующий протокол маршрутизации.

Примеры задания некоторых протоколов приведены ниже

```
set cmc [$ns mrtproto CtrMcast {}]     ; # задание типа centralized для
                                       ; # multicast маршрутизации на всех
                                       ; # узлах сети и создание ссылки cmc
                                       ; # на объект типа multicast protocol

$ns mrtproto DM $n1 $n2 $n3            ; # задание типа протокола DM
                                       ; # для узлов n1, n2 и n3
```

В режиме многоадресной передачи пересылка данных осуществляется заданной группе получателей. Для того, чтобы определенный узел-приемник смог участвовать в приеме информации, необходимо присоединить его к группе получателей данных. Добавить/удалить агент-приемник из определенной группы можно командами join-group {} и leave-group {}. Эти процедуры имеют два аргумента, определяющих соответствующий агент и адрес группы. Новый неиспользуемый multicast адрес может быть выделен процедурой allocaddr {} класса Node.

3.1. Пример модели многоадресной передачи в сети

Рассмотрим фрагмент модели простой сети, использующей многоадресную передачу.

```
set ns [new Simulator -multicast on]
set group [Node allocaddr]

set node0 [$ns node]
```

```
set node1 [$ns node]

set mproto DM
set mrthandle [$ns mrtproto $mproto {}]

set udp0 [new Agent/UDP]
$ns attach-agent $node0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0

$udp0 set dst_ $group

set rcvr0 [new Agent/Null]
$ns attach-agent $node1 $rcvr0
$ns at 0.1 "$ node1 join-group $rcvr0 $group"
$ns at 0.3 "$ node1 leave-group $rcvr0 $group"
```

Данный фрагмент скрипта определяет группу получателей с адресом определяемом [Node allocaddr]. Для всех узлов сети устанавливается протокол multicast маршрутизации DM (Dense Mode), что осуществляется командой $ns mrtproto $mproto {}. Создается источник данных и приемник, которым является определенная группа получателей ($group). Агент приемник rcvr0 присоединяется к группе получателей трафика в заданный момент времени 0.1 сек и покидает ее в момент времени 0.3 сек.

Каждый узел, не имеющий возможности принять трафик (например, не включенный в группу получателей), и на котором установлены агенты маршрутизации DM, при получении данных для определенной группы отсылает источнику уведомляющие сообщения о невозможности приема (пакеты типа 'prune'). Получение подобных сообщений источником переводят его в режим. “отсечения”('prune'). В этом режиме источник не может отсылать групповые данные узлу, пославшему 'prune' сообщение. По истечении некоторого времени ожидания (задается с помощью переменной PruneTimeout, например, 'DM set PruneTimeout 0.3' - время ожидания 0.3 сек), источник повторяет отсылку данных узлу приемнику.

3.2. Задание к работе

Создайте модель сети, показанной на рис. 4.11

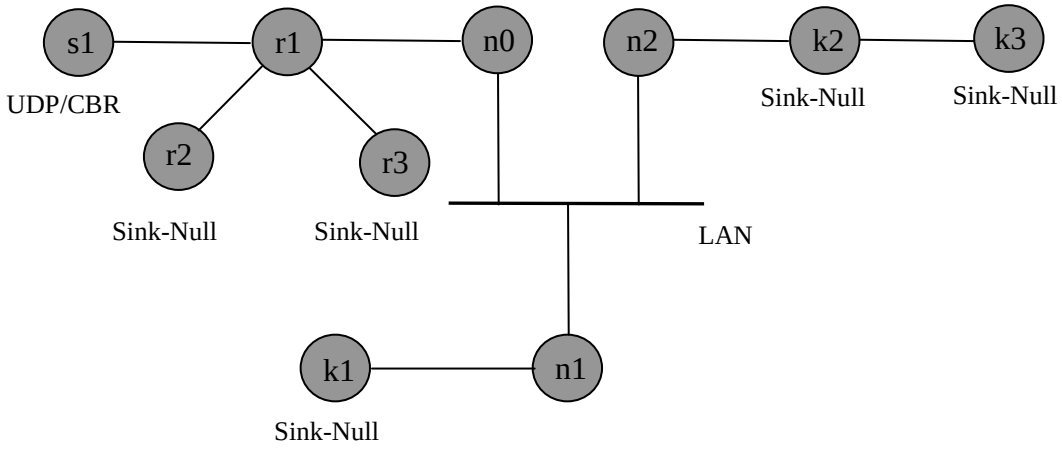


Рис 4.11.

Источник данных, находящийся в узле s1, представляет собой CBR агент, установленный поверх транспортного агента UDP. Задайте для него интервал между генерацией пакетов данных 0.005 сек. На узлах r2 – r3 и k1 - k3 находятся приемники трафика типа Null.

Локальная сеть содержит три узла n0 - n2 и имеет следующие характеристики:

механизм обработки очереди сетевого интерфейса - DropTail, ее максимальный размер - 50 пакетов, задержка распространения - 20 мсек, пропускная способность - 1Мбит/сек, тип агента канального уровня - LL, агенты Mac и физического уровня имеют тип Mac и Channel, соответственно.

Параметры всех остальных каналов связи одинаковы:

- Пропускная способность 1Мбит/сек;
- Задержка распространения 20 мсек;
- Механизм обслуживания очереди DropTail.

Установите в сети multicast протокол DM с величиной переменной PruneTimeout, равной 1 сек. В качестве адреса приемника данных UDP агента установите адрес группы [Node allocaddr].

Для отметки пакетов служебных сообщений можно установить, например, следующие цвета симулятора:

```
$ns color 30 purple
$ns color 31 blue
```

Задайте следующий порядок событий в сети:

Время (сек)	Событие
0.01	Старт CBR источника
0.15	Приемник данных r2 присоединяется к группе получателей
0.25	r3 присоединяется к группе получателей
0.3	k1 присоединяется к группе получателей
0.6	k2 присоединяется к группе получателей
0.9	k3 присоединяется к группе получателей
1.1	r2 покидает группу получателей
1.3	k3 покидает группу получателей
1.6	r3 покидает группу получателей
1.7	k2 покидает группу получателей
2.0	Окончание моделирования

Данные моделирования выводите в выходные файлы out.tr и out.nam. Скрипт сохраните в файле work6_2.tcl.

4. Программа работы

1. Изучите особенности моделирования локальных сетей в симуляторе . Ознакомьтесь с командами скрипта примера пункта 2.1.
2. Создайте файл work6_0.tcl с приведенным в п. 2.1 текстом скрипта и запустите его на моделирование. Обратите внимание на новые команды в скрипте, связанные с моделированием LAN.

3. Создайте скрипт в соответствии с индивидуальным заданием (п. 2.2), сохраните его в виде файла work6_1.tcl и запустите скрипт на моделирование. Проанализируйте результаты.
4. Ознакомьтесь с использованием протоколов multicast-маршрутизации и текстом примера (п. 3.1).
5. Выполните задание пункта 3.2 (файл work6_2.tcl). Обратите внимание на особенности работы протоколов маршрутизации, в случае использования LAN.

СПИСОК ЛИТЕРАТУРЫ

1. Т. Стернс Учимся моделировать, Сети, 1998, N5, [http://www.osp.ru/nets/1998/05]

2. Network Simulator- ns2. [http://www.isi.edu/nsnam/ns/]

3. Cohen R., Ramanathan S. TCP for High Performance in Hibrid Fiber Coaxial Broad-Band Access Networks. IEEE/ACM Transactions on Networking, 1998, v.6, N1, p15-29.

4. Fall K., Floyd S. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. Computer Communication Review, V.26, N.3, July 1996, pp.5-21. [http://www.aciri.org/floyd/papers.html]

5. XGraph - an animating plotting program [http://jean-luc.ncsa.uiuc.edu/Codes/xgraph]